1992-06

Version and variation control of a design database for a computer aided prototyping system

O'Loughlin, Michael Dennis

Monterey, California. Naval Postgraduate School

Approved for public release; distribution is unlimited

VERSION AND VARIATION CONTROL OF A DESIGN DATABASE FOR A COMPUTER AIDED PROTOTYPING SYSTEM

by
Michael Dennis O'Loughlin
Captain, United States Marine Corps
B.S., Kean College of New Jersey, 1983

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
June 1992

REPORT DOCUMENTATION PAGE

ORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
URITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution is unlimited
ASSIFICATION/DOWNGRADING SCHEDULE	
ORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S)
E OF PERFORMING ORGANIZATION iter Science Dept. Postgraduate School / 6b. OFFICE SYMBOL (if applicable) CS	7a. NAME OF MONITORING ORGANIZATION Naval Postgraduate School
RESS (City, State, and ZIP Code) rey, CA 93943-5000	7b. ADDRESS (City, State, and ZIP Code) Monterey, CA 93943-5000
E OF FUNDING/SPONSORING ANIZATION ial Science Foundation / 8b. OFFICE SYMBOL (if applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
RESS (City, State, and ZIP Code) ngton, D.C. 20550	10. SOURCE OF FUNDING NUMBERS: PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.

E (Include Security Classification)
ATION AND VERSION CONTROL OF A DESIGN DATABASE FOR A CAP SYSTEM (U)

SONAL AUTHOR(S)
ighlin, Michael D.

PE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
r's Thesis	FROM 06/91 TO 06/92	June 1992	346

PPLEMENTARY NOTATION The views expressed in this thesis are those of the author and do not reflect the official or position of the Department of Defense or the United States Government.

COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
D	GROUP	SUB-GROUP	Key Terms describing Your Work

STRACT (Continue on reverse if necessary and identify by block number)
e Computer Aided Prototyping System(CAPS) was created for rapidly prototyping real-time systems to deter-early in the software development life cycle wether system requirements can be met. The CAPS consists of sev-oftware tools that automatically generate an executable Ada model of the proposed system. This thesis describes evelopment of a design database (DDB) for CAPS. The DDB is an engineering database that contains all the nation related to a software prototype design. The DDB enhances the CAPS environment and the prototyping l by providing to the designer the capability to store, retrieve, view, edit, and control variation and versioning prototype components. This thesis describes the design, test and implementation of a tree structure variation ersion control method for supporting CAPS.

TRIBUTION/AVAILABILITY OF ABSTRACT NCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED
AME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE (Include Area Code) (408) 646-2735 / 22c. OFFICE SYMBOL CS/Lq

M 1473, 84 MAR 83 APR edition may be used until exhausted
All other editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

ABSTRACT

The Computer Aided Prototyping System (CAPS) was created for rapidly prototyping real-time systems to determine early in the software development life cycle wether system requirements can be met. The CAPS consists of several software tools that automatically generate an executable Ada model of the proposed system. This thesis describes the development of a Design Database (DDB) for the CAPS. The DDB is an engineering database that contains all information related to a software prototype design. The DDB enhances the CAPS environment and the prototyping model by providing to the designer the capability to store, retrieve, view and control variation and versioning of the prototype components. This thesis describes the design, test and implementation of a tree structure variation and version control method for supporting CAPS.

THESIS DISCLAIMER

Trademarks

ONTOS is a trademark of Ontologic, Inc.

TABLE OF CONTENTS

ACKNOWLEDGMENTS

Debbie, Connor and Shavonne:

For helping me get by the rough times and putting up with my long hours and absences. Debbie who kept telling me "I know you will finish." and Connor and Shavonne who were still willing to play with me no matter how bad a mood I was in.

Loretta O'Loughlin:

In the summer of 1970, my mother gave me one of her talks which was suppose to prepare me for the future. I will never forget the guidance she gave me that day. She said, "Mike, you know now that you are going into high school, it is not going to be like grammar school. They will offer a lot of different classes for you to take. If they ever offer a class about computers, I think you should take it. I think computers are going to be important someday."

Professor Williamson:

Your questions made me better understand what was going on.

Professor Luqi:

For the opportunity to do something big.

I. INTRODUCTION

A. THE NEED FOR A DESIGN DATABASE.

"The development of hard real-time and embedded software systems is an extremely complex and expensive process...Vast amounts of evolving data are created in the design of hard real-time systems" [Ref. 1:p. 1]. Management of this data is critical in computer aided design (CAD) environments [Ref. 2:p. 261]. There is a requirement for a database management system (DBMS) to be developed to meet the needs of systems being developed in CAD environments. Conventional DBMS's were designed for business applications and made available only the current value of the records stored in the database. The qualities of the currently available DBMS's do not meet the requirements of design engineers working in CAD environments. CAD systems require that the objects be stored and retrieved according to the needs of the design engineer. These objects must closely model the real world object they describe. The design of a system evolves over time, requiring the DBMS used in conjunction with CAD tools to store design data as it is viewed at different time periods during the system life cycle. This important feature which is a requirement of a design application is not available in commercial database management system (CDBMS).

Because the DBMS requirements of a design application are different from those of a business application, a new approach must be taken to meet the DBMS requirements supporting design applications. Though many CAD organizations have tried to integrate commercially available DBMS while developing their projects, they have found "that some of the important features which are required in the design application are not available in CDBMS" [Ref. 2:pp. 261-262]. Various solutions have been proposed to rectify the CDBMS problems of handling CAD databases. Berzins and Ketabchi [Ref. 3:p. 94] list four proposed solutions to the problem.

1. Develope a new DBMS, called a design DBMS (DDBMS), equipped with facilities required in the design application.

2. Enhance the current DBMS by adding new capabilities.

3. Build a layer of software on top of current DBMS's to compensate for their deficiencies.

4. Use a special purpose file manager that views the DBMS as a design application.

Of the four approaches listed by Berzins and Ketabchi the first is the approach selected for the creation of a design database for the Computer Aided Prototyping System (CAPS). As Berzins and Ketabchi [Ref. 2:p. 262] state "We have chosen to pursue this approach because we believe that the enhancements of CDBMS required in the second approach are too extensive to make this approach feasible, and that the third approach can not meet the flexibility and efficiency desired in CAD environments."

B. THE HISTORICAL TRAIL.

During the evolution steps executed on the objects of the DDB, a historical trail will be maintained so the design and decision history of the system being developed will be recorded.

The design history consists of the relationship between each version of the requirements and the corresponding version of the parts of the prototype [Ref. 4:p. 17]. This type of information will be useful when a decision is made to return to a previous version of the project. The historical trail will provide the information necessary to restore the corresponding parts of the prototype to the previous configuration.

C. THE OBJECT ORIENTED APPROACH.

The need for a design database, as stated in section A, is the inherent difference between the DBMS requirements of a design application and those of business applications. The object oriented approach is "based on object oriented data models rather then record oriented data models...The object oriented data model (ODM) will increase the

productivity of the design systems by providing modeling facilities which mirror the designs logical view of the data" [Ref. 2:p. 261].

There are three basic parts of an ODM:

1. Object: convenient aggregations of information describing real world objects.

2. Properties: functions which model characteristics of objects and relationships among them.

3. Operations: Change the state of an object, enforce constraints and provide an application oriented interface to the database [Ref. 2:p. 261].

The development of object oriented technology (object oriented languages, object oriented programs and object oriented data models) has allowed designers to develop DDBMS meeting the requirements of design applications. This object oriented approach allows the use of off the shelf Objected Oriented DBMS which can be further developed to possess the enhanced capabilities needed for design applications.

Object oriented database management systems (OODBMS) provide several features not available in other database types. Two of the major advantages of an OODBMS are the increased modeling power and the ability to closely align and store CAD objects to reflect real-world relationships. This supports the definition of a design database as defined by Katz [Ref. 5:p. 379], "a (large) collection of objects that together describe an artifact being designed... 'Objects' are usually packages of data and manipulation procedures." The term object as used by Katz, contains the three parts of the ODM listed above.

Over the past several years, several OODBMS have been developed. The impact of these systems on the software development process are just being felt. The object oriented approach represents a true paradigm shift [Ref. 6:p. 386].

In laying out the conceptional design of the design database Douglas [Ref. 1:p. 8] clearly states what the OODBMS must provide.

"An object-oriented database management system OODBMS) must provide persistence, concurrency, recovery, transaction management, authorization, and security. An OODBMS is an objected-oriented system and as such it must also provide the following capabilities:

- Objects
- Active data
- Abstraction
- Extensibility

An OODBMS should provide application_oriented capabilities such as:

- Version and configuration control for CAD applications.
- Dynamic creation of classes.
- Recursive classes, multiple inheritance, and extensive tool interface capabilities.
- Support for multimedia objects, distributed environments, and graphs.

An object-oriented DBMS is one that supports persistency, values, an extensible set of data structures, an extensible set of operations, and abstractions."

D. THEORY OF THE DESIGN DATABASE.

The purpose of a design database (DDB) is the management of system project data, so that the system components can be stored and retrieved according to the needs of the design engineers.

The DDB is an independent software system in its own right. It is capable of being a stand alone system which can manage the design data of any system under development. In the context of this thesis, the DDB is a tool contained within CAPS and supports the prototype systems being designed by CAPS. Although the DDB supports all the prototype systems being designed by CAPS "there will be only one instance of the design database for each project" [Ref. 1:p. 14]. For example, each design engineer will feel that the system they are working on has it own design database to manage the design data of their system.

The physical location of the data will be a concern of the actual database management system. The design database being developed uses the ONTOS Object Database. ONTOS, a distributed object database management system, decides where to physically store the data [Ref. 11:p. 3] and makes access to the design data transparent to the user.

The DDB manages the design data by manipulating collections of design data identified as nodes. Within CAPS, a node is made up of PSDL text, Ada source code, graphic representation data and a postscript representation of the node. The PSDL text is separated into specification and implementation components.

The DDB is a hierarchial storage structure of nodes. Each node represents a different component. This hierarchial structure is a tree structure consisting of atomic and composite

nodes. "Each level of the tree is created by the decomposition of the parent node. The decomposition process is complete when all leaf nodes are atomic" [Ref. 1:p. 19].

The nodes of a DDB are time sensitive. The nodes will version and create new variations depending when they are accepted into the design database. This topic will be discussed further in chapter two, section two.

The design database is accessible to the user through a highly developed user interface or a low level command line interface. Since the commands issued to the DDB will have the same effect no matter which platform they are issued from, the DDB is independent of the interface used for accessing the design data.

The major functions of the DDB within CAPS are:

1. Store the levels of a PSDL program in a hierarchial format by specification.
2. Retrieve the levels of a PSDL program in a hierarchial format by specification for review or editing.
3. Create and insert new levels of a PSDL program in a hierarchial format by specification.
4. Generate the entire PSDL program [Ref.1:p. 26].

Douglas [Ref.1:p. 27] gives an excellent example describing the process in which the four listed functions must work.

"To construct a prototype, the PSDL specification of the root operator is entered. At this point the DDB would create a root node. Assuming the root node is composite, the node would decompose into children operators. The DDB would create child nodes for each decomposition. The decomposition would continue until all leaf nodes are atomic... the functions of retrieving nodes, parent-child relationships, and deleting nodes will be required. The tree will be traversed and the entire PSDL program produced once all leaf nodes are atomic."

II. BACKGROUND

A. COMPUTER AIDED PROTOTYPING SYSTEM.

The Computer Aided Prototyping System (CAPS) is a collection of computer-aided software tools designed to support prototyping of complex software systems, such as control systems with hard real time constraints. Prototyping can be carried out manually, but because of the cost in time and effort, manual prototyping produces limited benefits. The aim of CAPS is to strengthen the prototyping strategy by allowing the system designer to develop and modify the prototype automatically to meet the changing requirements of the user. The prototype goes through an evolutionary process as do all software systems. As the system proceeds through its evolutionary steps, the tools provided by CAPS will aid the designer in modifying the prototype to meet new requirements established by the user.

The main components of CAPS are a special prototyping language and a set of software tools [Ref. 4:pp. 14-15], Figure 1.

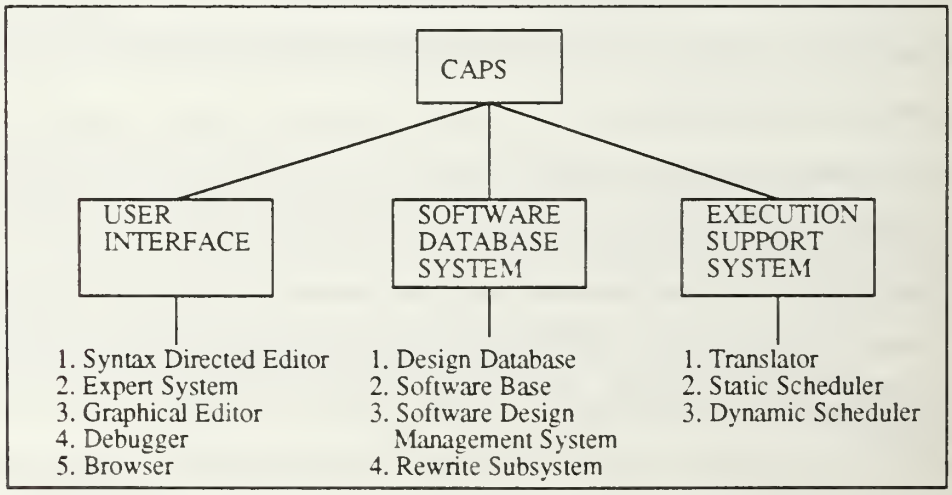


Figure 1. Computer Aided Prototyping System Tools.

The main components of CAPS are the three tool subsystems: The User Interface, Software Database System and the Execution Support System. The special prototyping

language that allows communication between the tools is the Prototype System Description Language (PSDL). PSDL integrates the tools and provides the designer with a uniform conceptual frame work and a high level description of the system [Ref. 4:p. 15]. PSDL enhances the power of the designer to modify the prototype by providing the following capabilities:

- Modularity
- Adaptability
- Abstraction
- Requirements tracing

The User Interface is composed of five software tools which provide the designer with the ability to enter requirements and design information, display the results of a prototype execution, select the different prototype capabilities to demonstrate, and propagate changes throughout the prototype. The user interface contains the:

- Syntax Directed Editor
- Graphic Editor
- Expert System
- Debugger
- Browser

The syntax directed editor allows for the entry and modification of syntactically correct PSDL code. The designer will use the syntax directed editor to edit the PSDL code generated by the graphic editor, and enter PSDL descriptions and text of prototype design. The text portions of the prototype are not represented in the graphic editor. PSDL files are identified by the suffixes ".spec.psdl" and ".imp.psdl".

The graphic editor supports a graphic view of the prototype. It allows the designer to design an augmented data flow diagram of the prototype under development. The graphic objects displayed in the editor represent PSDL descriptions of the prototype. Skeletal PSDL code will be generated from the data flow diagrams designed using the graphic editor. The graphic editor generates files with ".spec.psdl", ".imp.psdl", ".ps" and ".graph" suffixes.

The expert system gives the user, who is unfamiliar with PSDL, the capability to examine the prototype by generating english text from PSDL.

The browser allows the designer to view and retrieve the reusable PSDL and Ada components from the software database.

The debugger allows the designer to interact with the execution support system. From the debugger the designer can execute the prototype, display results and gather statistics about a prototypes behavior and performance. The debugger contains two components: a debugger for the static scheduler and one for the dynamic scheduler. The static scheduler debugger processes errors while attempting to create a static schedule and the dynamic debugger processes errors that occur while the prototype is executing.

The Software Database System contains four software tools.

- the Design Database
- the Software Base
- the Software Design Management System
- the Rewrite Subsystem

The system is structured to allow for the reuse of prototype designs and reusable software components during the development of the prototype. Reuse is the key motivation for the software database system. The benefits of reusing software include:

- improved software quality and maintenance
- increased programmer productivity and efficiency
- lower development costs [Ref. 7:p. 94]

The requirements established for the software database system provide for the following capabilities:

- to store PSDL designs and software components
- to retrieve designs and software components for editing
- the CAPS user interface should integrate with the software database system in such a way that the software database management system is transparent to the prototype designer [Ref. 7:p. 97].

The design database is a repository for the different variations and versions of PSDL prototype descriptions for all software projects developed using CAPS. Each prototype being developed under CAPS will appear to have its have its own design database.

The software base contains PSDL descriptions and Ada source code for all reusable software components.

The software design management system maintains the prototype design history. This system is used to locate reusable software components from the software base and retrieve the versioned system prototypes from the design database.

The Execution Support System is made up of three tools:

- the Translator
- the Static Scheduler
- the Dynamic Scheduler

The translator generates high level code from the PSDL prototype which binds the reusable components from the software base to the executable prototype [Ref. 4:p. 17]. The translator will generate Ada source code. The Ada source code files generated will be identified by a ".a" suffix. The purpose of the translator is to produce an Ada translation of a PSDL prototype description. The translator performs lexical analysis of the internal textual representation of a PSDL system prototype, parses the prototype description and constructs an abstract syntax tree. After evaluating the tree attributes, it then constructs Ada source code utilizing PSDL abstract data types.

The static scheduler schedules time constraints that are assigned to PSDL operations to ensure that all time constraints are met during execution.

The purpose of the dynamic scheduler is to coordinate the execution of operators and their debuggers during execution.

B. THE ROLE OF THE DESIGN DATABASE IN RAPID PROTOTYPING.

The CAPS Design Database is constructed of C++ and ONTOS classes. Each class serves a separate and unique function in the prototyping process. This section explains the concepts associated with each class and how they relate to the prototyping process.

1. Classes in the Database Design.

a. Prototype.

The term prototype used in this thesis denotes "a complete executable model of selected aspects of a proposed system [Ref. 4:p. 13]."

A prototype is the highest level class of the design database. Systems under development in CAPS cannot be stored in the design database unless a prototype name has been entered into the database under which the system will be stored. A prototype can consist of numerous versions of a system under development. Each version of the system can be a "complete executable model" or a partially developed system that was identified as not worthy of further development. An instance of a prototype class represents all the versions being developed under that particular prototype name.

b. Versioned Object.

"An object is a software component that is subject to change... A version is an immutable snapshot of an object. Versions have unique identifiers. New versions can be created but versions cannot be modified after they are created [Ref. 8:p. 917]." A versioned object can be composite or atomic. A composite object can be decomposed into two or more objects. An atomic object cannot be decomposed into other objects.

When an object, either atomic or composite, is inserted into the design database initially, it is assigned the control number sequence "variation 1, version 1".

Each object inserted into the database will be assigned a version number. The version numbers will be assigned sequentially as the object evolves from one version to the next. The exception to this is when a new variation is created. Since versioning is sequential

a new variation is created when a versioned object can not version using the next number in the sequence. For example, if variation 1, version 3 of object A is modified but variation 1, version 4 of object A already exists, a new variation will be created. The variation number is also assigned sequentially. If the highest variation number assigned is 1, then variation 2, version 1 will be assigned to the new object A.

Figure 2 gives an example of an atomic and a composite object. The composite object Operator_B decomposes into the atomic objects Op_B1, Op_B2 and Op_B3.

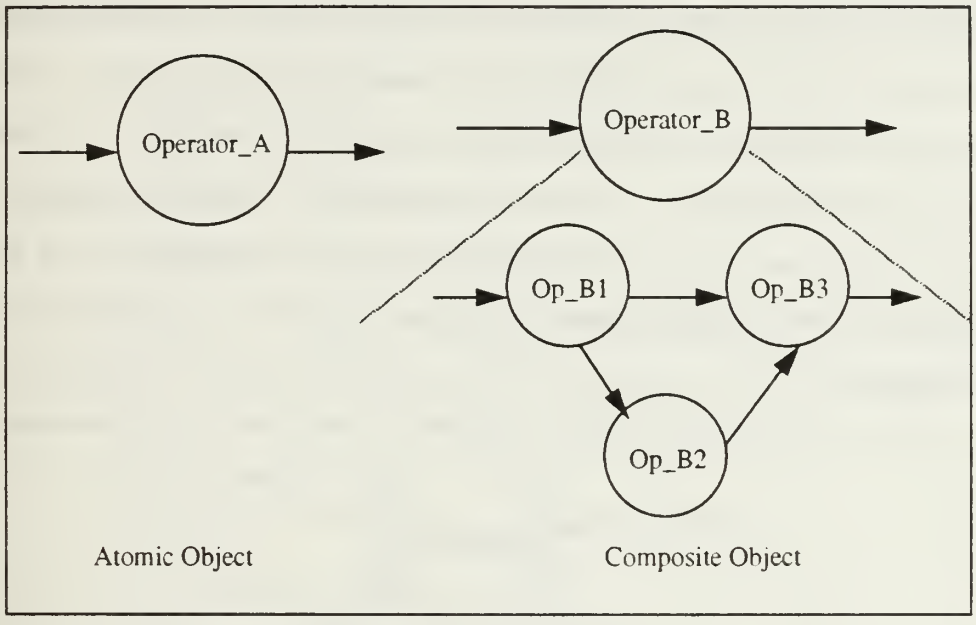


Figure 2. Atomic and Composite Objects.

The initial insertion of the objects to the DDB will result in the following control number sequence assignments:

Operator_A	Variation 1, Version 1
Operator_B	Variation 1, Version 1
Op_B1	Variation 1, Version 1
Op_B2	Variation 1, Version 1
Op_B3	Variation 1, Version 1

These numbers are a part of an object's properties and determine its uniqueness forever. Each versioned object is considered to be created once it has been inserted into the DDB. Once created an versioned object cannot be modified. Therefore, for each subsequent retrieval of any versioned object, the designer will only receive a copy of the versioned object contained in the DDB. If the designer modifies his copy of the versioned object, the versioned object will version when inserted back into the DDB. Retrieving Operator_A from the DDB, modifying it and inserting it back into the DDB will cause the creation of a new version. The new object will be identified as Operator_A "variation 1, version 2". When Operation_B is retrieved from the DDB the decomposed objects, Op_B1, OP_B2 and Op_B3, which make up Operator_B will also be retrieved. If Operator_B is modified it will version upon insertion into the DDB. If its decomposed objects are not modified they will not version. Figure 3 shows an example in which OP_B1a is added to Operator_B and generates an additional output data flow. This additional data flow changes Operator_B, but has no effect on any of the decomposed atomic objects.:

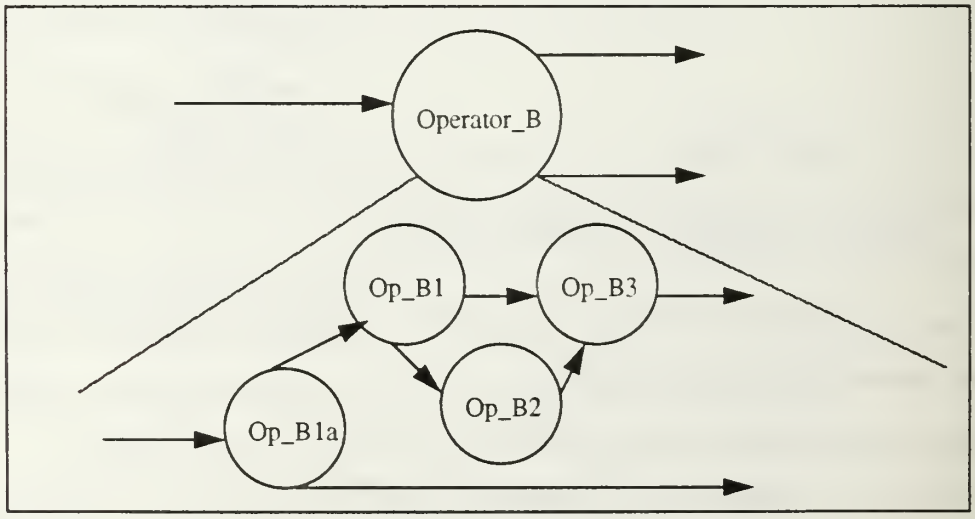


Figure 3. Operator_B Versions.

The input declarations of to Op_B1 have not been modified, so no modifications are required of Op_B1. After the modified objects have been inserted into the DDB they will be identified as follows:

Operator_B	Variation 1, Version 2
Op_B1a	Variation 1, Version 1
Op_B1	Variation 1, Version 1
Op_B2	Variation 1, Version 1
Op_B3	Variation 1, Version 1

Another modification that could have taken place is the addition of a state variable to Op_B2, as shown in Figure 4. In this case OP_B2 and Operator_B will version and all the other objects will remain the same. When inserted in the DDB after the previous modifications, the objects would be assigned the following control number sequences:

Operator_B	Variation 1, Version 3
Op_B1	Variation 1, Version 1
Op_B2	Variation 1, Version 2
Op_B3	Variation 1, Version 1

As described in Figure 3 the top level operator has been modified and therefore it will version when inserted into the DDB. In Figure 4 only a decomposed atomic

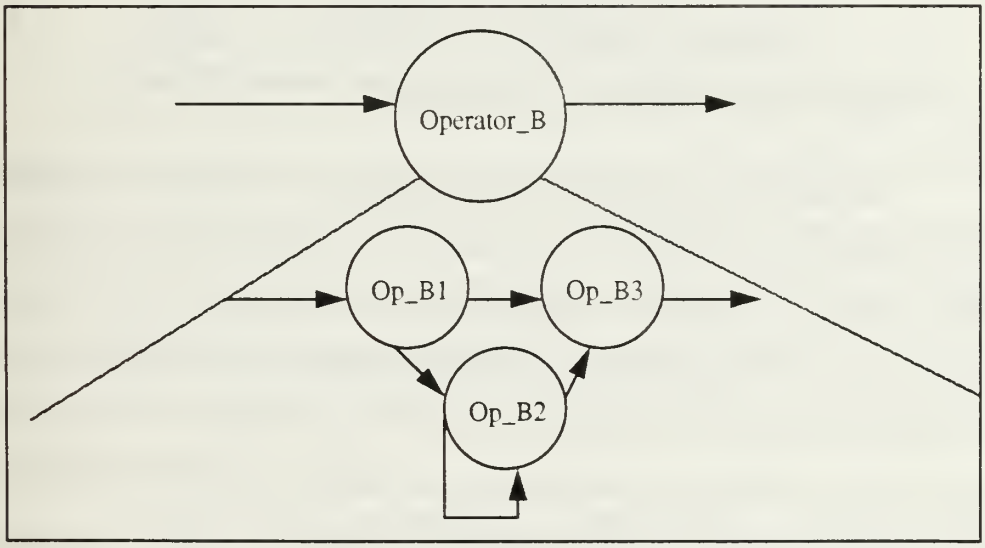


Figure 4. Op_2 Versions.

object is modified. The Op_B2 object is therefore versioned when inserted into the DDB. The versioning of the atomic object OP_B2 propagates up the tree versioning each

successive parent until the root node has versioned. Only those nodes for which Op_B2 is a direct descendent will version.

Objects are not limited to decomposing only to a second level as described in the examples above. Op_B3 could be a composite object and not just an atomic object. If Op_B3 were a composite object, then each of Op_B3's decomposed objects (composite or atomic) would version as described. If Op_B3 were a composite object Operator_B would decompose as shown in Figure 5.

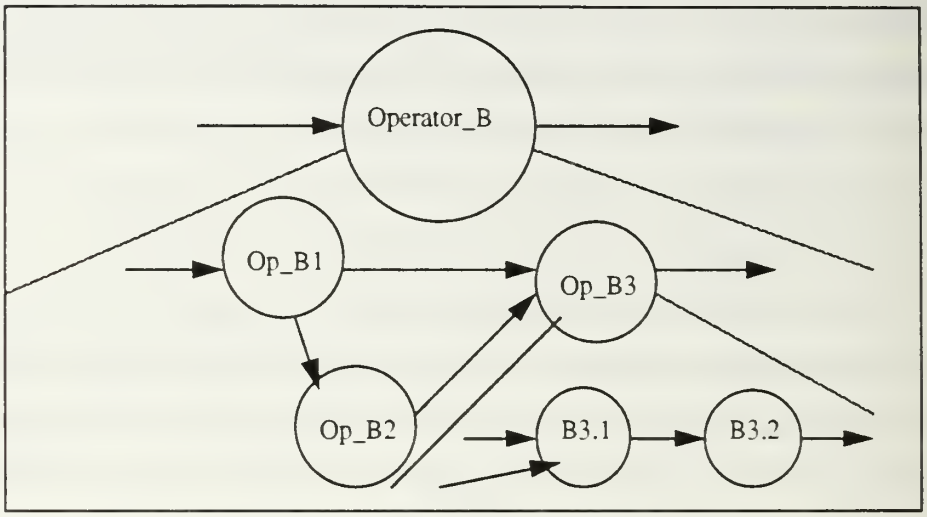


Figure 5. Decomposition to Composite and Atomic Objects.

An instance of a versioned object represents one version of an operator in the prototype system being developed. The versioned object can be atomic or composite. It is a operator node in the tree which represents the decomposition of the system under development. Each versioned object is uniquely identified by the object's name and the objects variation numbers. The variation number is used to separate the Versioned objects because the version numbers repeat themselves for each new thread. In each instance of a thread the versioned object is unique because it possesses a version number that is a minimum distance of 1 from the version number of any other versioned object.

c. Thread.

An instance of a thread corresponds to a variation in the control number sequence. The term thread denotes a C++ class used to manage the paths of development of a versioned object. The term variation is the equivalent to thread; therefore variation will be used in this document where thread is used in the written C++ code. A variation is an alternate path of development. A variation contains a totally ordered sequence of versions of an object [Ref. 8:p. 918] with strictly increasing creation times. Each version of an object belongs to exactly one variation. A new variation will be created if and only if an version about to created is not the latest version of the variation.

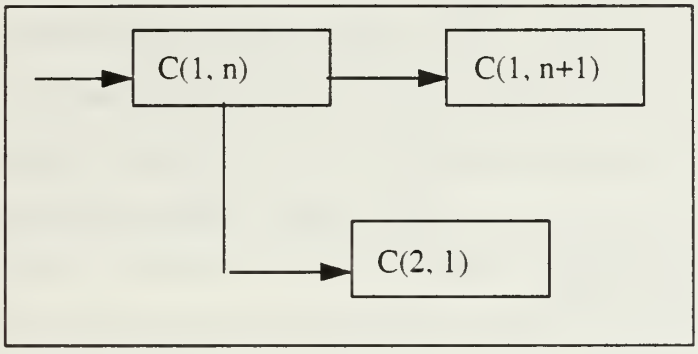


Figure 6. Creation of a New Thread (Variation).

Figure 6 shows a new variation being created. When an object is retrieved from the DDB and modified it can version in one of two ways. Depending on the intention of the modifications being made, the system will version along the same variation path or version and create a new variation path, in other words, a new parallel line of development. In figure 6 the notation C(a, b) is used to identify the versioned object. C(a, b) is read as version b of variation a of object C [Ref. 9:p. 26]. Modifying C(1, n) will create a new variation when the following occurs:

(1) C(1, n) is retrieved from the DDB when C(1, n+1) already exists.

(2) C(1, n) is then modified by the designer and reinserted into the DDB.

Since C(1, n+1) already exists and C(1, n) has been modified, this new versioned object is reinserted as C(2, 1). This versioned object C(2, 1) is the beginning of

a new variation whose beginning versioned object will be C(2, 1). When C(2, 1) is retrieved and modified, it will follow the same versioning scheme as all versioned objects, and will be reinserted as C(2, 2).

Each thread is uniquely identified by the versioned object's name and variation number.

d. Configuration.

A configuration is a selected version of a sub system under development. The designer can select any version and place it under a configuration name. The configuration contains the entire subtree of the versioned object. The versioned object is a complete sub system selected by the designer which contains certain aspects of the system at a point in time during the systems evolution. The configuration allows the designer to more effectively deal with and identify selected versions of a system under development.

The configuration represents a version of the system under development and therefore can not be modified.

e. Component.

A component can be either composite or atomic. The top level view of composite components will give a general overview of a system or object. As the component decomposes the view of the objects become more detailed. A composite component can be viewed as a collection of related parts [Ref. 9:p. 2]. Large systems, like those which will be developed using CAPS, are made up of top level composite components with several layers of other composite components between them and their lowest level atomic components.

Within the DDB a component contains a list of Text Objects which make up the composite or atomic operator. The parts of an atomic component are a subset of ".a", ".ps", ".spec.psdl", "imp.psdl" generated by the graphic editor. The component is at the base of the class hierarchy. The component is a list of pointers to the actual source text, and graphic representation data and PSDL code stored in the DDB. The pointers contained in

the list point to the files created by the graphic editor and modified by the syntax directed editor. These files contain the following information:

Postscript: Postscript description language required by the graphic editor.

Graph: A file containing shapes and geometric point information required by the graphic editor.

Implementation: A file containing either Ada source code or a PSDL decomposition. This file is used by the syntax directed editor, the dynamic scheduler and the translator.

Specification: A file containing PSDL code. This file is used by the syntax directed editor, the dynamic scheduler and the translator.

Source: Ada code. A file containing the source code for the component and prototype [Ref. 10:pp. 33-34].

A composite component contains information concerning the component of the next lower level, wether next lower component be composite or atomic.

f. Text Object.

Text object contains the name of the file and the actual source text for the operator. The text can be any of the five types pointed to by the component class.

2. Connections Between Classes.

The classes of the design database are not independent of each other. Each C++ class contains ONTOS based classes that connect that class to other C++ classes or ONTOS

classes which make up the database schema. Figure 7 shows a pictorial representation of these connections without listing all the fields of the database schema.

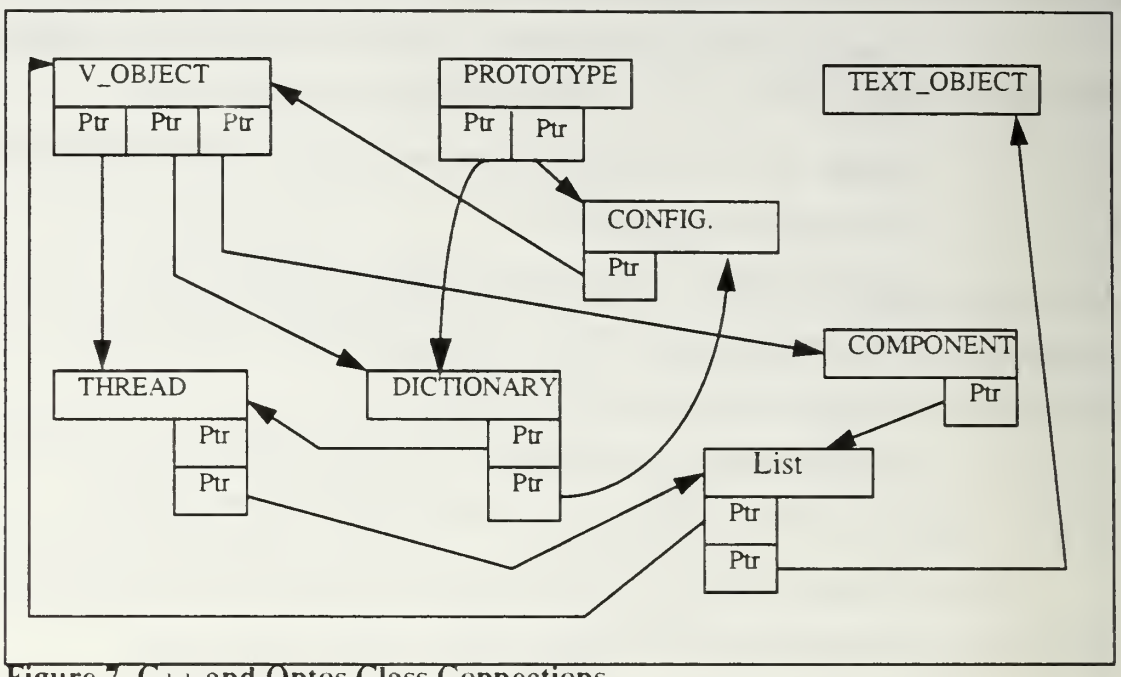


Figure 7. C++ and Ontos Class Connections.

These connections between classes are described in detail in chapter III section C.

C. OTHER ATTEMPTS AT ENGINEERING DESIGN DATABASES.

"The goals of configuration management include recording the development history of evolving systems, maintaining the integrity of such systems, and aiding the management of the systems in guiding and controlling their evolution [Ref. 8:p. 918]."

Until recently most attempts at providing support for configuration management consisted of a combination of manual and administrative procedures. This type of configuration management was labor and time intensive and prone to errors. Attempts at automating the process led to storing many versions of the same document and keeping up to date mechanically derived software.

One attempt at maintaining the integrity of configurations is addressed in the module interconnection languages. "This purpose of the module interconnection language is to record the interdependencies between components of a system [Ref. 8:p. 918]." This

includes keeping track of functional specifications and structural and syntactic properties. The module interconnection language only deals with specifications and programs. Another attempt [Ref. 14] supports upward compatibility, strict compatibility and implementation compatibility. Implementation compatibility is a weaker restriction than strict compatibility which allows the substitution of specifications within particular contexts. Maintaining mechanically-derived software is another effort that has been under taken [Ref. 15].

The shortcomings of these efforts is that they only deal with the most current version and not all the versions leading up to the current version [Ref. 8:p. 919].

Luqi [Ref. 8:pp. 919-921] describes the Model of Software Manufacture and the Model of Software Evolution which support configuration management.

The aim of the Model of Software Manufacture is to aid in the managing of mechanically derived objects, with reducing the number of objects that must be rebuilt in response to a change. Also it attempts to estimate the computing cost of implementing the change.

The Model of Software Evolution attempts to integrate software evolution and configuration management. This work is based on the ANSI/IEEE standard on Software Configuration Management [Ref. 8:p. 921].

III. IMPLEMENTATION OF A DESIGN DATABASE FOR CAPS

A. OBJECTIVES.

The design database currently being used to support the CAPS environment was developed by Dwyer and Lewis [Ref. 10]. This design database will be identified as the "current" database while the design database developed in this thesis will be called the "new" database.

The current Design Database does not meet all the requirements necessary to support CAPS. The requirements that it does support need to be enhanced or redesigned while other functions need to be developed in order to fulfill the requirements established by CAPS.

The schema for the design database needs to be redesigned in order to meet the requirements to support variation and version control. The current design database structure stores versions in a linear progression. When an object is retrieved from the database, modified and reinserted, it versions but it will never create a new variation. If version 2 of object A is retrieved from the current database and modified, and version 3 of object A already exists, version 2 will be reinserted into the current database as version 4 of object A. Since here is no link between (object A version 2), and (object A version 4) it can not be determined that version 4 came from version 2. The new database must provide the functionality to support variations as described in chapter 2.

All operations that access the design database need to be redesigned. The current operations only access objects on a single development path. One of the requirements for the CAPS design database is to possess the ability to create and store alternate paths of development. Each new path is a new variation of an existing object. Each variation contains sequentially numbered versions of an object.

The current database creates new versions of an object based solely on the time stamp of the object. This allows objects to version when no actual changes have taken place. Each object needs to be evaluated based on changes in content and not on an external time stamp.

A historical trail operation is required to trace the development of any object back to its original root object. This requirement is necessary because the new database design will be based on a tree structure with variations being capable of branching off any object. The need to know the evolution of an object is important when making decisions concerning future development.

After an object is reinserted into the current database, a copy of the object still remains in the designers working directory. When an object has been successfully inserted into the database, the copy remaining in the designers working directory needs to be removed.

The current design database does not fully support the ability to choose selected versions of different objects and place them together to develop a new system. This requirement is necessary to avoid rebuilding a section of a system that has already been designed and tested.

Currently objects which the designer wants for view only, are retrieved from the database and stored in the designers working directory in read only mode. This requires that they later be removed by the user in order to retrieve the object for editing. This is an unnecessary requirement placed on the designer. Access to all objects in the design database, for view or for modification should require no additional work on the part of the designer other then issuing commands to insert or retrieve objects.

The addition of variations to the development path makes much of the current design database code obsolete and dangerous. Inadvertent execution of obsolete code can cause system failure, data corruption, or numerous other problems without the user being aware that a problem has occurred. This code needs to be studied, identified and removed while valid code is left intact.

B. METHODOLOGY.

All design methodologies share a similar path of development at a certain time during the project life cycle. Every project goes through the activities of system analysis, design, and implementation, even if the names given the activities are different for each methodology used. The classical project life cycle, also know as the "waterfall life cycle" has stages known as Analysis, Program Design, Coding, Testing, and Operations. The semi-structured life cycle is composed of activities named Analysis, Structured Design, and Top-Down Implementation. The Structured Design activity is further decomposed into Codify Functional Specification, Derive Structure Chart, Design Module, and Package Design. The structured project life cycle contains activities such as Analysis, Design, Implementation, Acceptance Test Generation, Quality Assurance and Installation.

The activities listed above are not all of the activities of the project life cycle methodologies named above. They only represent a part of the system life cycle in which the work being done on the system is similar in nature no matter which methodology is used to develop the system. The areas that they represent are the design, coding, testing and implementation activities. The actual implementation of these individual activities is what separates the different methodologies [Ref. 12:pp. 77-101].

Yourdon [Ref. 12:p. 94] points out that more then one activity can occur at a time in the structured project life cycle. Depending on the number of activities occurring at any one time it will be known as a radical or conservative implementation of the project life cycle.

Douglas [Ref. 1] conducted the survey and requirements analysis to establish the needs of the design database. Dwyer and Lewis [Ref. 10] expanded on the requirements analysis established by Douglas. They also designed, implemented and tested some of the functions established by the requirements analysis. A review of their thesis shows a structured life cycle approach was taken.

The issues dealt with in this thesis were the design of a new database structure, that being the redesign of the way in which the objects are stored and retrieved. We have designed a new database schema to support the new database structure. Because the

attribute keys for the objects were altered, the insertion and retrieval of objects from the database also had to be redesigned.

The comparison of the objects based on content had to be implemented. Versioning of objects based on the value of a time stamp is unacceptable.

The project life cycle methodology used for this thesis is best described as a moderately radical structured approach [Ref. 12:p. 95]. The design of new and redesign of existing structures along with implementation and testing all had to occur at the same time. Each new function and modification of existing functions had to be tested to evaluate their effect on other functions and objects. All modifications to existing functions had to be noted and reflected and placed as updates in the users manual. Of the nine activities that Yourdon [Ref. 12:p 89] lists as part of the structured project life cycle, four of them were simultaneously being implemented during the development of the new design database These four activities are design, implementation, acceptance test generation and quality assurance.

C. DESIGN DECISIONS AND JUSTIFICATIONS.

1. Database Schema.

The database schema of the new design database is made up of user defined and pre-defined ONTOS C++ classes [Ref. 11]. The manipulation of these C++ classes allows the user to add, delete or modify objects stored in the design database. The design database is made up of six user defined classes. Within these six classes there are two ONTOS defined classes which help support the tree structure necessary to maintain variations and versions. The user defined classes are derived from the ONTOS class Object. The six user defined classes are:

Component

Configuration

Prototype

Text_Object

Thread

Versioned_Object

The ONTOS classes used within in the user defined classes are:

Dictionary

List

The relationship between these classes is a spider web of pointers, Figure 7.

The Prototype class is at the top level of the hierarchial design structure. This class has pointers to Configuration and Text_Object user defined class objects and to an ONTOS class Dictionary object. The Configuration object holds the default configuration for the prototype. There may be many configurations attached to a prototype but the Configuration pointer points to the latest configuration attached. All the other configurations attached to the prototype are accessed through the Dictionary object. The default (latest) configuration has its own pointer but a pointer to this configuration is also placed in the Dictionary class. The Dictionary class contains pointers to all the Configurations attached to the prototype. A Text_Object object contains the text description of the Prototype. Figure 8 gives a pictorial representation of the Prototype class and its member classes. Other data fields are

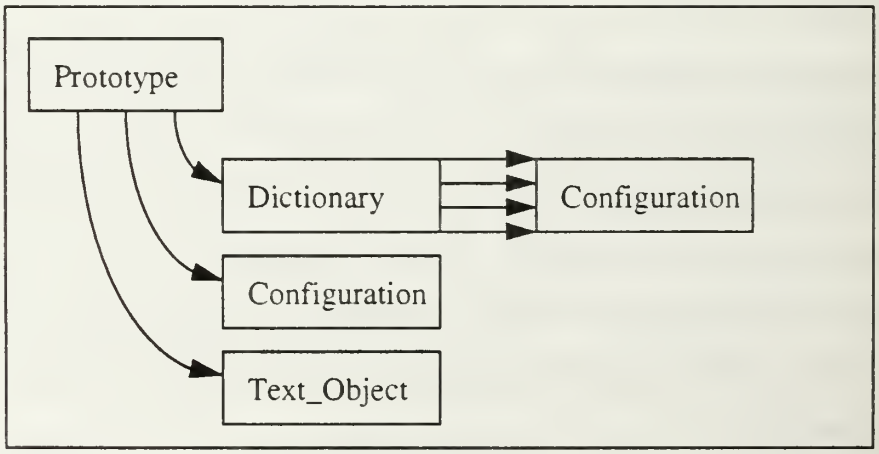


Figure 8. Prototype class and internal classes.

also contained within Prototype and are used for accounting and historical information. The Prototype class has no other classes pointing to, or accessing its data.

The Configuration class points to Versioned_Object and Text_Object objects. Configuration points to two instances of Text_Object. The first instance contains the configuration description provided by the designer. The other instance contains log entries. Log entries are additional information concerning the makeup of the configuration. The Versioned_Object pointer points to the versioned object which is the root node of a tree or sub tree to be identified by a unique configuration name. The unique configuration name can identify an entire system or any subtree of a system. The Versioned_Object identified by the configuration name can be accessed for view or editing. If a Versioned_Object is modified when retrieved using the configuration name, the Versioned_Object will version but the Configuration name will not reflect the new version.

For example, variation 1, version 3 of object A is identified by the unique configuration name config_1. Variation 1, version 3 of object A is retrieved from the database using the name config_1. After the modifications have been completed and config_1 is reinserted into the database, the configuration name, config_1 will still point to variation 1, version 3 of object A. Meanwhile a new object has been created, and is identified as variation 1, version 4 of object A.

All other data fields contained in the Configuration class are used for accounting and historical purposes. Configuration is pointed to and accessed from Prototype. Figure 9 gives a pictorial representation of the Configuration class and the class member objects contained within it.

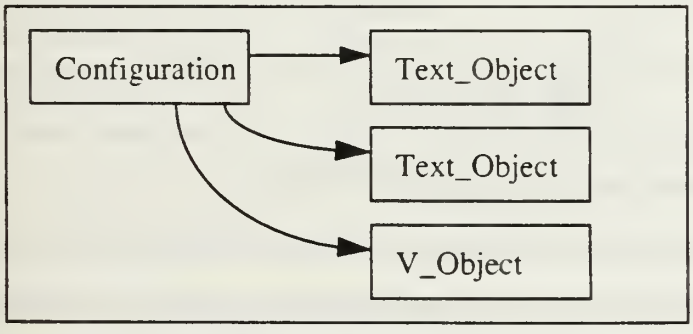


Figure 9. Configuration and its internal classes.

The Component class, figure 10, contains only one element, the ONTOS defined List class. The List object contains a list of pointers to instances of Text_Objects. The Text_Objects contain information detailing wether the component is composite or atomic.The Component class is a container class for Text_Objects. A Component class can contain any number of Text_Objects.

The Text_Object class, figure 10, is at the bottom of the hierarchial class structure. The Text_Object class contains two character string pointers. All ".a", ".graph", ".spec.psdl", ".imp.psdl" ".as", log and description entries are contained in Text_Objects. One character string pointer points to a file name, the other to the file containing the actual text. Because of required access to text files, Text_Object is pointed to by Prototype, Configuration, Versioned_Object and List classes. Each instance of Text_Object, depending which class points to it, contains a different type of text.

The Versioned_Object class, figure 11, is a class which points to, or is accessed by almost every other class in the database. The Versioned_Object class is the crossroads of the design database structure. The Versioned_Object points to other Versioned_Objects, Text_Objects, Components, a Thread, a List and two instances of Dictionary classes. It is pointed to by Versioned_Object, List, Configuration and Dictionary classes.

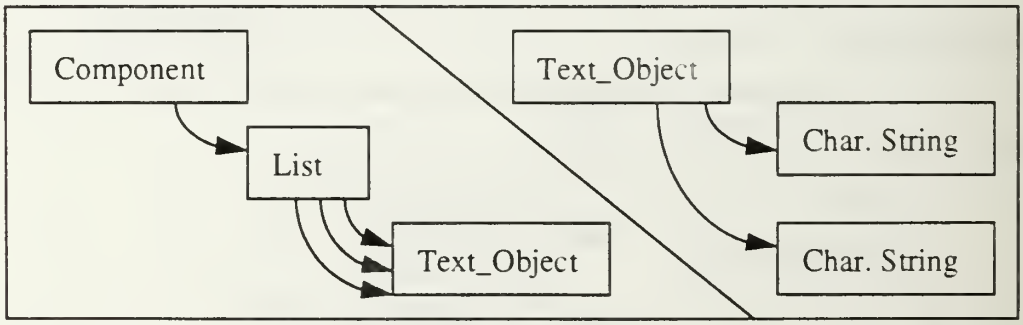


Figure 10. Component and Text_Object Classes.

One Dictionary object contains the pointers to the first Versioned_Object of each new Thread. The other Dictionary object contains pointers to all the Threads which contain Versioned_Objects which versioned off this particular Versioned_Object. The List class contains the pointers to all the Versioned_Objects which are children of this

Versioned_Object. The Thread pointer points to the Thread that the Versioned_Object is attached.

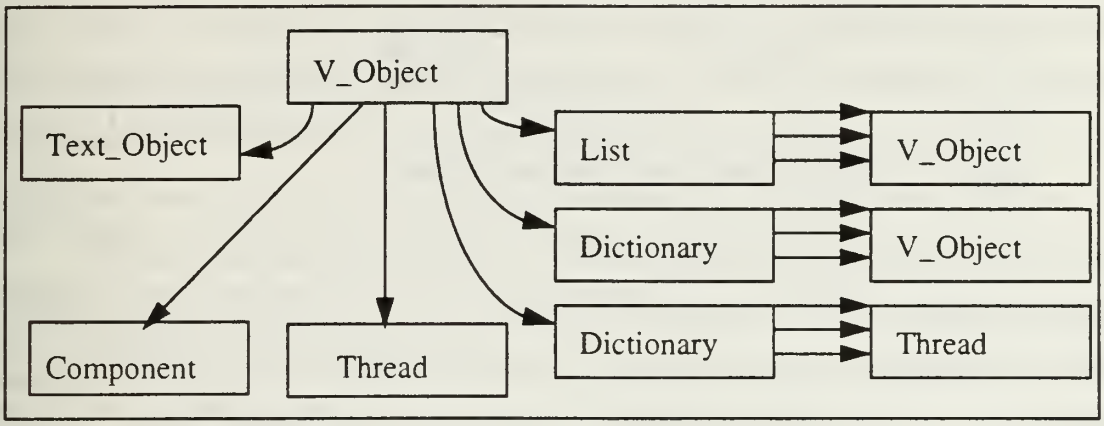


Figure 11. V_Object class and internal classes.

The Component pointer points to the List object, which contains pointers to Text_Objects associated with this Versioned_Object. The Versioned_Object pointer points to the Versioned_Objects parent if it is not the root node, otherwise the pointer is null. The Text_Object points to a description of the Versioned_Object provided by the designer.

The Thread class, figure 12, is similar to the Component class for the reason that it is a container class for Versioned_Objects. The Thread class contains accounting data, historical data and a List class object. The List object points to Versioned_Objects attached to the Thread. Thread is pointed to by Versioned_Object. The thread class is the C++ equivalent of a variation.

2. New Versions and New Variations.

Each time an object is modified by the designer it will version. The new object created will be a copy of the object retrieved from the database incorporating the modifications made by the designer. The new object can take one of two separate paths of development. The new object can become a new version, and continue on the current path of development or the new object could become the first version of a new variation. A modification of an object at any level will propagate changes up the tree to the root object.

Because a node on a subtree has changed the functionality of the entire tree has also changed and therefore versioning of the entire tree must take place.

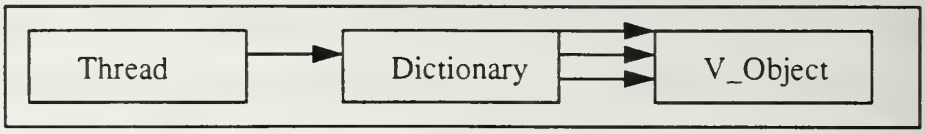


Figure 12. Thread class and its internal classes.

The current design database creates a new version based on the value of the objects time stamp. The time stamp is updated for an object each time the object is accessed while in the designers working directory. This does not mean that the object has been modified, just accessed. When inserting the object into the design database the current database compares the time stamp of the object being inserted against the time stamp of the object in the database. If the value of the time stamps are different, a new version of the object will be created. This process does not take into consideration the actual contents of the object. Versioning is dependent upon an external time stamp variable.

The new design database compares the contents of the objects and creates a new version if the actual contents of the objects are different. Since an atomic object, at a minimum, consists of between one and five Text_Objects contained in the Component class, each Text_Object is compared individually. If the contents of any one of the possible five Text_Objects is different from the contents of the corresponding Text_Object, of the object stored in the database, then the object will version. The new design database then will insert into the database the new object from the designer's working directory.

Each versioned object is contained on a thread. A thread is the C++ class representation of a variation. When a new object is created, (if no new variation is created) it is made unique by creating it with a version number one higher then the previous object. The name given the object is the same as the operator name assigned the object in the graphic editor.

A new variation will be created when an object being inserted into the database has been modified and there exists an object with a higher version number, on the same

thread, of the object being inserted. The modified object will become the first versioned object of the new variation. The thread is given a unique name by concatenating the object name and the variation number. The thread that contains the operator c3i_system.sensor_interface, variation 1, version 2 is actually identified in the database as c3i_system.sensor_interface_1. All the versions that make up variation 1 of c3i_system.sensor_interface are contained in the thread c3i_system.sensor_interface_1.

If the conditions for the creation of a new variation are met, the new variation will be identified in the database as c3i_system.sensor_interface_2. The thread has been made unique by appending the variation number to the object name. This thread/variation will contain only versions of the c3i_system.sensor_interface object. The c3i_system.sensor_interface objects can be assigned version numbers that were assigned in previous thread/variations because they are attached to a different and unique thread/ variation.

3. History Trail.

The historical trail as implemented in the new design database is extremely simple.

The current design database had no need for a historical trail. Since it handled only one variation, the history of each version is simple. The current version is assumed to have been created from the previous version. With the inclusion of variations the historical trail of object development becomes more complicated. Each new variation created requires a link back to its parent object.

The new design database now lists the versions of any selected versioned object in reverse chronological order from the given to the root object.

4. Objects in the Working Directory and the Concerns of the Design Database.

The new design database places object control files in the designers working directory. These files in no way interfere with the designers ability to retrieve objects from the database, but they do provide control data to the database when reinserting an object

into the database. When the command is issued to retrieve an object from the database the operator name along with the variation and version numbers provided on the command line identify the root object to be placed in the designers working directory. Any children (objects), subtrees attached to the root object identified on the command line will also be retrieved from the database and placed in the designer's working directory. For each object placed in the designer's working directory a database control file is written to the directory also. The control file contains the variation and version numbers of the object that has been retrieved. The designer will not see this control file when accessing the objects via the user interface or the command line. The control data file name is made up of the complete object name prefixed by "ddbCtrlData." The complete object name is a concatenation of all the object names as they decompose to the object. For example, the c3i_system, c3i_system.sensor_interface and c3i_system.sensor_interface.add_sensor_data will have the following control files:

ddbCtrlData.c3i_system

ddbCtrlData.c3i_system.sensor_interface

ddbCtrlData.c3i_system.sensor_interface.add_sensor_data

When the objects are being reinserted into the database each object has its control file read just prior to being reinserted. The DDBMS uses the information contained in the control file to locate the object in the database. Once located, the necessary comparisons can be conducted to establish wether a new object needs to be created.

The control files are removed from the directory after the appropriate action has been taken to create or not create a new object.

Under the current design database, copies of all object inserted into the database remain in the designer's working directory. If the designer wants, at some later time, to retrieve the same object or a different variation/version of the object, the copies have to be removed. If all the copies are not removed, some objects will be retrieved from the database and others will not. Once the database attempts to write an object to a directory with an object of the same name already existing in that directory, the write fails and the DDBMS

stops retrieving objects from the database. The objects that have been successfully written to the directory are now locked by the database and the complete object tree has not been retrieved from the database. The designer has only one option, remove all the objects just retrieved from the working directory along with the duplicate object. Issue a release lock sub tree command to release the locks of the objects just retrieved and initiate a new retrieval.

The new design database removes all copies of objects from the designers directory after successful insertion of the object into the database.

If for some reason a copy of an object remains in the designer's directory and then the object is later retrieved the DDBMS will ask the designer if they want the old copy to be overwritten. If the designer responds yes, overwrite, the DDBMS overwrites the object in the designers directory and continues to retrieve objects from the database. If the designer responds no then the same procedure as the current database must be followed.

5. Viewing and Editing Objects in the Working Directory.

Depending on the parameter passed to the database, any versioned object retrieved will be either be read only (view) or editable. If the r or R option is used, the versioned object retrieved will be placed in view mode. The designer will be able to access the system and all its components, but will not have the capability to alter any of the object's contents. The objects retrieved using the view option will be sent to standard out. The user interface will then redirect the output from the database to an appropriate directory. The directory where the system components will be stored will not be the designers working directory. Under the current implementation of the design database, the system selected to be viewed is placed in the designers working directory with file permission set for read only. This setup later causes additional work for the designer. When the same system is retrieved from the database for purposes of editing, the existing read only files will not allow the database to store the retrieved data in the working directory.

This causes the retrieval to terminate. The designer then has to delete each object before being able to retrieve the system for editing.

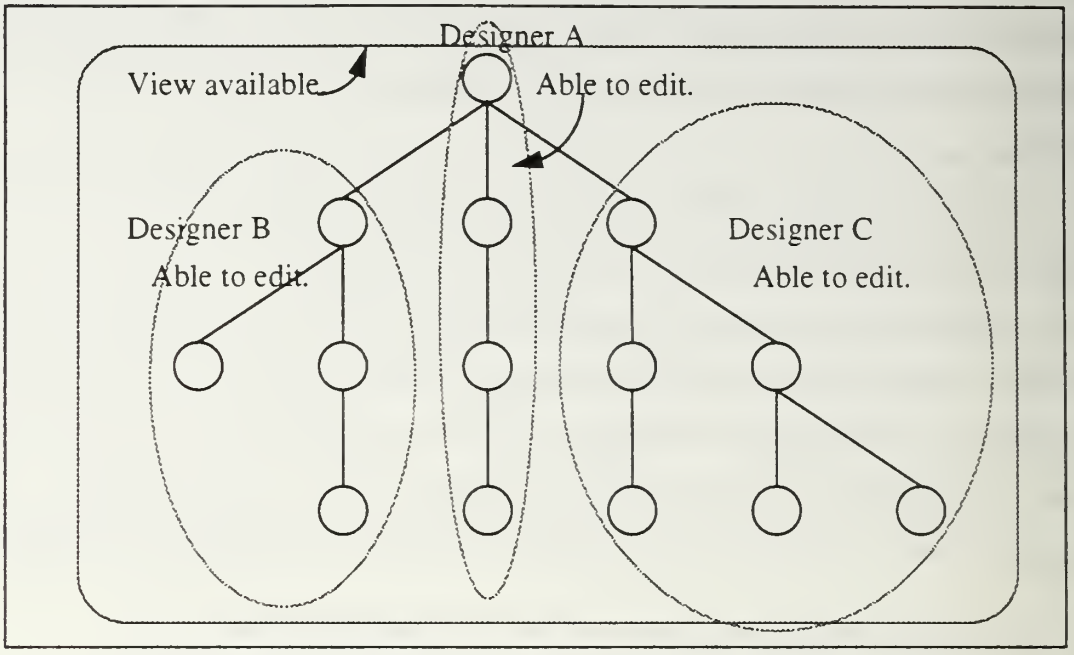


Figure 13. Separate Workers on the same system.

When retrieving an object for editing the w or W option is used. The object retrieved is placed in the designers working directory. The object and all its components are now available for editing or viewing. The designer now has full control over the object retrieved from the design database. If two or more designers want to work on different sections of the same object tree, in their own directories, each can retrieve from the design database the section of the object they will work on. When a tree or subtree of an object is retrieved from the database it is locked and then available for view only to all other designers. The other designers working on the system can retrieve the locked sections of the object for view only and have a complete view of the object while actually working on only a part of the object. Figure 13 show how a system could be split up among the designers.

6. Dangerous Obsolete Code.

The necessity to modify, enhance and redesign the existing design database in order to bring it up to the established requirements of the CAPS system, entailed the detailed study of existing code. Although some of the functions are reusable as written, most of them required extensive modifications or needed to be completely rewritten.

As code was modified or added, an examination of the code in the immediate area was reviewed for potential problems. Required changes were made where necessary, deletions were made if necessary and test of all changes were conducted. All code found to be obsolete was removed. As much as possible obsolete code has been identified and removed.

D. TESTING AND TEST RESULTS.

Frakes [Ref. 13:p. 125] lists three definitions of software testing. The definition which best describes the testing performed on the new design database is "any activity aimed at evaluating an attribute or capability of a program or system and determining that it meets its required results."

Four stages of testing are expected to be traversed before a system can be accepted. They are:

unit testing

integration testing

system testing

acceptance testing

The new design database has undergone three of the four test stages.

Unit testing, the testing of a single function, was conducted on each function written or modified. Each function was accessed via the command line to ensure that the function performed according to its specifications. No function was tested independently, but required the interaction of other functions, to call or be called, to complete the actual execution of the command.

Integration testing was conducted when new functions were required to be designed and implemented. The new functions were grouped together, according to their functionality, and tested to check for mutual interference or data corruption.

The acceptance test was conducted in the form of a demonstration of the capabilities of the new design database using systems developed by other designers.

The system test, where the system is integrated with the user interface and tested, has not been conducted.

Because of the amount of code already written and the large number of required changes that had to be implemented, regression testing was only test strategy that was acceptable to use.

Whenever code is modified or inserted to correct a problem or enhance the system one of four things may occur:

1. The problem is fixed or the enhancement is successful.
2. The problem was not corrected or the enhancement is not as successful as planned.
3. The problem is fixed or the enhancement is successful, but a new problem has been introduced.
4. The problem was not corrected or the enhancement is not successful, and a new problem has been introduced [Ref. 13:p. 137]

Since only the first result is acceptable, tests were conducted over and over again each time a modification was implemented. Regression testing was not implemented for each change made but rather for each group of changes and additions made for a localized functionality. A shell script was constructed to aid in this consuming and cumbersome testing procedure.

The shell script and test results are listed in appendix C.

IV. RECOMMENDATIONS AND CONCLUSIONS

A. SUMMARY

Although the main goals of this thesis were met, the implementation of variation and version control, there are numerous smaller problems that exist which are annoying to the designer and detract from the capabilities of the design database.

B. RECOMMENDATIONS FOR FUTURE WORK

The system suffers from unhandled exception errors. All exceptions should be caught and the system gracefully shut down or have other appropriate action. Examples of errors that shut the system down are:

Inserting an object into the database with no object in the directory.

Using variation and version numbers that do not exist for an object.

If the design database successfully terminates but an error was detected prior to termination the DDBMS will still commit the objects to the database. For example, the retrieval of objects is terminated if an object of the same name is in the designer's directory. The designer has the option to overwrite the object if they choose to but if they choose not to, the objects already retrieved are locked. The designer's directory now contains all objects successfully retrieved prior to the error. These objects are only a part of the object's tree. The database is updated to lock all the objects successfully retrieved. Since the retrieval of the entire object tree was not successful the DDBMS should not update the design database.

The dependency on control files in the designer's working directory should be removed. All the required information for an object is in the database. If the root object is known the each of the children objects can be identified.

The ability to access objects and their data via versioned object commands and configuration commands is redundant. The configuration commands can be removed and all access could be through versioned object commands.

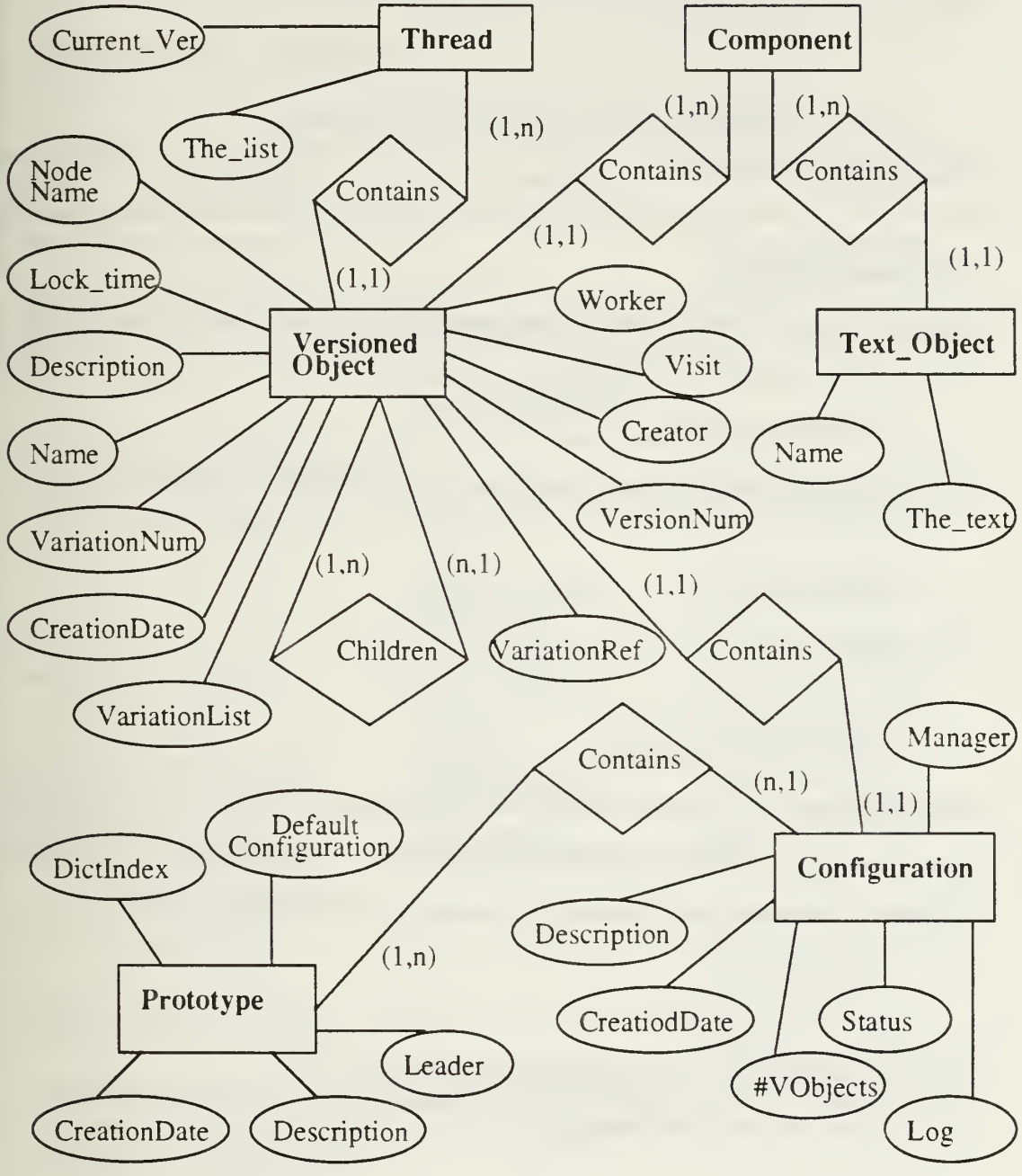
The DDB schema should be reworked to simplify insertions and take advantage of subclasses and inheritance.

If a object identified by a configuration is modified, the configuration should identify the new object. This should occur if the object was accessed by the configuration name.

Configurations appear to have limited usefulness. The design of the DDB and its role in CAPS should be reviewed to determine wether the concepts and structures related to configurations can be remover from the DDB.

APPENDIX A

A. ENTITY-RELATIONSHIP (ER) SCHEMA DIAGRAM



B. DATABASE SCHEMA

THREAD

Current_Version, {The_list}

TEXT_OBJECT

The_file_name, The_text

PROTOTYPE

Leader, DictIndex, CreationDate, {ConfigurationList}, Description, DefaultConfig

CONFIGURATION

Status, Manager, CreationDate,VobjectNum, Log, Description, Versioned_Object

COMPONENT

{Text_object_list}

VERSIONED_OBJECT

VariationNum, VersionNum, CreationDate, LockTime, NodeName, Creator, Worker, Visit, Last_Op_Checkin, Description, {ThreadPtr}, {ComponentPtr}, {ChildPtr}, ParentPtr, {VariationList}, NumberOfVariations, {VariationReference}

Multivalued attributes are shown between set braces { }.

There are no composite attributes.

APPENDIX B

COMMAND LINE COMMANDS

A. PROTOTYPE COMMANDS

Table 1: PROTOTYPE COMMANDS

Tag	Description	Output	Command Line[a]
PIP	Insert Prototype		ddb <db> pip <p>[b] [leader] [description][c]
PLN	List Names	Names (one per line)	ddb <db> pln
PLL	Long List	Name, Default Config. Default VOBJECT	ddb <db> pll
	Retrieval Commands:		
PDS	Dump Summary	Date Created Leader Default Config Description	ddb <db> pds <p>
PRD	Retrieve Date	Date Created	ddb <db> prd <p>
PGL	Get Leader	Leader	ddb <db> pgl <p>
PGC	Get Configuration	Default Config	ddb <db> pgc <p>
PGD	Get Description	Description	ddb <db> pgd <p>
	Update Commands:		
PUL	Update Leader		ddb <db> pul <p> <"new leader">
PUD	Update Description		ddb <db> pud <p> <filename>

a. <> Angle brackets are required parameters. [] Square brackets are optional parameters.
b. <db> db = Ontos Database name. <p> p = Prototype name.
c. Passed in as a filename or string.

B. CONFIGURATION COMMANDS

Table 2: CONFIGURATION COMMANDS

Tag	Description	Output	Command Line[a]
CIC	Insert Configuration		ddb <db> cic <p> <c>[b][manager] [description][c]
CLN	List Names	Names	ddb <db> cln <p>
CLV	List Default VOBJECT	Name, Version	ddb <db> clv <p> <c>
CLO	List Operators	Operator Name,[d] Version	ddb <db> clo <p> <c>
CLL	Long List Default VOBJECT's Children	Node[e] Name, Version	ddb <db> cll <p> <c>
	Retrieval Commands:		
CDS	Dump Summary	Date Created Date Changed Manager Default VOBJECT/ Version Description	ddb <db> cds <p> <c>
CDA	Get Date Created	Date Created	ddb <db> cda <p> <c>
CGM	Get Manager	Manager	ddb <db> cgm <p> <c>
CGD	Get Description	Description	ddb <db> cgd <p> <c>
CGL	View Log[f]	Log Entries	ddb <db> cgl <p> <c>
	Update Commands:		
CUN	Update Name		ddb <db> cun <p> <c> <new name>
CUM	Update Manager		ddb <db> cum <p> <c> <new_name>
CUD	Update Description		ddb <db> cud <p> <c> <filename>

Table 2: CONFIGURATION COMMANDS

Tag	Description	Output	Command Line[a]
CPL	Post Log		ddb <db> cpl <p> <c> <filename>
CRL	Release Lock		ddb <db> crl <p> <c>
CAA	Update VOBJECT's subtree		ddb <db> caa <p> <c>
CAO	Attach VOBJECT to CONFIGURATION		ddb <db> cao <p> <c> <vobject> [variation] [version]
	Extraction Commands:		
CDT	Dump VOBJECT subtree	file(s)	ddb <db> cdt <p> <c> R/W[g]

a. <> Angle brackets are required parameters. [] Square bracketsare optional parameters.
b. <db> db = Ontos Database name. <p> p = Prototype name. <c> c = Configuration name.
c. Passed in as a filename or string.
d. Operator name containing explicit path information for determining location (level) in heirarchical data structure.
e. Node name contains no reference to location (level) in heirarchical data structure
f. Log file is read only. Log can only be updated. Additional log entries are appended to the bottom of the current log.
g. R for View Only (Read). W for edit (Write).

C. VERSIONED COMPONENT COMMANDS

Table 3: VERSIONED COMPONENT COMMANDS

Tag	Description	Output	Command Line[a]
VAA	Add VOBJECT and Subtree	Confirmation	ddb <db> vaa <p> <v>[b] [variation] [version]
VLO	List Operators	VOBJECT Name,[c] Variation, Version	ddb <db> vlo <p> <v> [variation] [version]
VLL	Long List VOBJECT's Children	VOBJECT Name,[d] Variation, Version	ddb <db> vll <p> <v> [variation] [version]
VLP	Long List VOBJECT's Parent and Siblings	VOBJECT Name Variation, Version	ddb <db> vlp <p> <v> [variation] [version]
	Retrieval Commands		
HTT	History Trace	VOBJECT Name Variation, Version	ddb <db> htt <p> <v> [variation] [version]
VDS	Dump Summary	Date Created Creator Worker Lock Time Description	ddb <db> vds <p> <v> [variation] [version]
VDD	Get Date Created	Date Created	ddb <db> vdd <p> <v> [variation] [version]
VGL	Get Lock Time	Date, Time Locked	ddb <db> vgl <p> <v> [variation] [version]
VGV	Get Versions	Variation/Version Number (one per line)	ddb <db> vgv <p> <v>
VGD	Get Description	Description	ddb <db> vgd <p> <v> [variation] [version]
	Update Commands:		

Table 3: VERSIONED COMPONENT COMMANDS

Tag	Description	Output	Command Line[a]
VUD	Update Description	[e]	ddb <db> vgd <p> <v> <filename>[f] <variation> <version>
VRO	Release Lock Operator		ddb <db> vro <p> <v> [variation] [version]
VRS	Release Lock Operator and Subtree		ddb <db> vrs <p> <v> [variation] [version]
	Extraction Commands:		
VGP	Get Postscript	Display Postscript File	ddb <db> vgp <p> <v> [variation] [version]
VGG	Get Graphics	Display Graphics File	ddb <db> vgg <p> <v> [variation] [version]
VGI	Get Implementation	Display Implementation File	ddb <db> vgi <p> <v> [variation] [version]
VGC	Get Specification	Display Specification File	ddb <db> vgc <p> <v> [variation] [version]
VGS	Get Source	Display Source Code File	ddb <db> vgs <p> <v> [variation] [version]
VDF	Dump Operator Atomic[g]	file(s)	ddb <db> vdf <p> <v> <R/W> [variation] [version]
VDT	Dump Operator and Subtree	file(s)	ddb <db> vdt <p> <v> <R/W> [variation] [version]

a. <> Angle brackets are required parameters. [] Square brackets are optional parameters.
b. <db> db = Ontos Database name. <p> p = Prototype name. <v> v = Vobject name.
c. Operator name containing explicit path information for determining location (level) in heirarchical data structure.
d. Node name contains no reference to location (level) in heirarchical data structure.
e.
f. File containing description.
g. Atomic is considered a node containing .ps .graph .imp.psdl .spec.psdl and .a file(s).

APPENDIX C

A. TEST SCRIPT FILE.

This test was run with the aid of a shell script. The shell script is listed in section C. All design database command line commands begin with the word "main". The result of each command is displayed after each command line. If the line following a command line is blank, then there was no output for that particular command. The results of the command are accessible from others commands. For example, the first command executed is "pip", insert prototype name. There are no output results for this command to displayed, so the line following the command will be blank. The actual commands displayed are produced using echo, and only represent the command executed. For that reason any string in single quotes ('') should actually be placed in double quotes ("").

START COMMAND LINE TEST

main oloughln pip c3i_system_prototype 'Michael D. O'Loughlin' descp

main oloughln pln
prototype_c3i_system
c3i_system_prototype

main oloughln pll
prototype_c3i_system
c3i_system_prototype

main oloughln pgd c3i_system_prototype
This is a test file added to the design database for example purposes.
Any thing can be added to it, and nothing can be expected back.

main oloughln pud c3i_system func_description

main oloughln pgd c3i_system_prototype
This file is a test file so that all commands that access or affect the description can be tested.

All Prototype operations have been tested successfully.

Adding to configuration description.

All Prototype operations have been tested successfully, EXCEPT :

Removing operators CRL and CAA from cofiguration command.

Adding to Versioned component description.

main oloughln prd c3i_system_prototype
Mon Jun 8 14:50:24 1992

main oloughln pgl c3i_system_prototype
Michael D. O'Loughlin

main oloughln pul c3i_system_prototype 'Larry Williamson'

main oloughln pgl c3i_system_prototype
Larry Williamson

main oloughln pds c3i_system_prototype
Mon Jun 8 14:50:24 1992
Larry Williamson
This file is a test file so that all commands that access or affect the description can be tested.

All Prototype operations have been tested successfully.

Adding to configuration description.

All Prototype operations have been tested successfully, EXCEPT :

Removing operators CRL and CAA from cofiguration command.

Adding to Versioned component description.

main oloughln cic c3i_system_prototype c3i_config_1 'Michael D. O'Loughlin' descp

main oloughln cln c3i_system_prototype
c3i_config_1

main oloughln cda c3i_system_prototype c3i_config_1
Mon Jun 8 14:50:54 1992

main oloughln cgm c3i_system_prototype c3i_config_1
Michael D. O'Loughlin

main oloughln cgd c3i_system_prototype c3i_config_1
This is a test file added to the design database for example purposes.
Any thing can be added to it, and nothing can be expected back.

main oloughln cum c3i_system_prototype c3i_config_1 'Marty Shoppenheimer'

main oloughln cgm c3i_system_prototype c3i_config_1
Marty Shoppenheimer

main oloughln cud c3i_system_prototype c3i_config_1 func_description

main oloughln cgd c3i_system_prototype c3i_config_1
This file is a test file so that all commands that access or affect the description can be tested.

All Prototype operations have been tested successfully.

Adding to configuration description.

All Prototype operations have been tested successfully, EXCEPT :

Removing operators CRL and CAA from cofiguration command.

Adding to Versioned component description.

main oloughln cpl c3i_system_prototype c3i_config_1 'This is the first Post to c3i_system Log'

main oloughln cgl c3i_system_prototype c3i_config_1

Mon Jun 8 14:51:18 1992

This is the first Post to c3i_system Log

main oloughln cpl c3i_system_prototype c3i_config_1 'This is the second Post to c3i_config_1 Log'

main oloughln cgl c3i_system_prototype c3i_config_1

Mon Jun 8 14:51:18 1992

This is the first Post to c3i_system Log

Mon Jun 8 14:51:23 1992

This is the second Post to c3i_config_1 Log

main oloughln cln c3i_system_prototype
c3i_config_1

main oloughln cds c3i_system_prototype c3i_config_1
Mon Jun 8 14:50:54 1992
Marty Shoppenheimer

This file is a test file so that all commands that access or affect the description can be tested.

All Prototype operations have been tested successfully.

Adding to configuration description.

All Prototype operations have been tested successfully, EXCEPT :

Removing operators CRL and CAA from cofiguration command.

Adding to Versioned component description.

main oloughln vaa c3i_system_prototype c3i_system
c3i_system

main oloughln pll
prototype_c3i_system
c3i_system_prototypec3i_config_1

main oloughln vlo c3i_system_prototype c3i_system
c3i_system.comms_interface.convert_to_text_file 1 1
Wed Dec 31 16:00:00 1969
c3i_system.comms_interface.decide_for_archiving 1 1
Wed Dec 31 16:00:00 1969

c3i_system.comms_interface.extract_tracks 1 1
Wed Dec 31 16:00:00 1969
c3i_system.comms_interface.forward_periodic_report 1 1
Wed Dec 31 16:00:00 1969
c3i_system.comms_interface.make_routing 1 1
Wed Dec 31 16:00:00 1969
c3i_system.comms_interface.parse_input_file 1 1
Wed Dec 31 16:00:00 1969
c3i_system.comms_interface.prepare_periodic_report 1 1
Wed Dec 31 16:00:00 1969
c3i_system.comms_interface 1 1
Wed Dec 31 16:00:00 1969
c3i_system.comms_links 1 1
Wed Dec 31 16:00:00 1969
c3i_system.navigation_system 1 1
Wed Dec 31 16:00:00 1969
c3i_system.sensor_interface.analyze_sensor_data 1 1
Wed Dec 31 16:00:00 1969
c3i_system.sensor_interface.prepare_sensor_track 1 1
Wed Dec 31 16:00:00 1969
c3i_system.sensor_interface 1 1
Wed Dec 31 16:00:00 1969
c3i_system.sensors 1 1
Wed Dec 31 16:00:00 1969
c3i_system.track_database_manager.add_comms_track 1 1
Wed Dec 31 16:00:00 1969
c3i_system.track_database_manager.add_sensor_track 1 1
Wed Dec 31 16:00:00 1969
c3i_system.track_database_manager.filter_comms_track 1 1
Wed Dec 31 16:00:00 1969
c3i_system.track_database_manager.filter_sensor_track 1 1
Wed Dec 31 16:00:00 1969
c3i_system.track_database_manager.monitor_ownship_position 1 1
Wed Dec 31 16:00:00 1969
c3i_system.track_database_manager 1 1
Wed Dec 31 16:00:00 1969
c3i_system.user_interface.display_tracks 1 1
Wed Dec 31 16:00:00 1969
c3i_system.user_interface.emergency_status_screen 1 1
Wed Dec 31 16:00:00 1969
c3i_system.user_interface.get_user_inputs 1 1

Wed Dec 31 16:00:00 1969
c3i_system.user_interface.manage_user_interface 1 1
Wed Dec 31 16:00:00 1969
c3i_system.user_interface.message_arrival_panel 1 1
Wed Dec 31 16:00:00 1969
c3i_system.user_interface.message_editor 1 1
Wed Dec 31 16:00:00 1969
c3i_system.user_interface.status_screen 1 1
Wed Dec 31 16:00:00 1969
c3i_system.user_interface 1 1
Wed Dec 31 16:00:00 1969
c3i_system.weapons_interface 1 1
Wed Dec 31 16:00:00 1969
c3i_system.weapons_systems 1 1
Wed Dec 31 16:00:00 1969

main oloughln vll c3i_system_prototype c3i_system
comms_interface 1 1
comms_links 1 1
navigation_system 1 1
sensor_interface 1 1
sensors 1 1
track_database_manager 1 1
user_interface 1 1
weapons_interface 1 1
weapons_systems 1 1

main oloughln vlo c3i_system_prototype c3i_system.comms_interface
c3i_system.comms_interface.convert_to_text_file 1 1
Wed Dec 31 16:00:00 1969
c3i_system.comms_interface.decide_for_archiving 1 1
Wed Dec 31 16:00:00 1969
c3i_system.comms_interface.extract_tracks 1 1
Wed Dec 31 16:00:00 1969
c3i_system.comms_interface.forward_periodic_report 1 1
Wed Dec 31 16:00:00 1969
c3i_system.comms_interface.make_routing 1 1
Wed Dec 31 16:00:00 1969
c3i_system.comms_interface.parse_input_file 1 1
Wed Dec 31 16:00:00 1969
c3i_system.comms_interface.prepare_periodic_report 1 1

Wed Dec 31 16:00:00 1969

main oloughln vll c3i_system_prototype c3i_system.comms_interface
convert_to_text_file 1 1
decide_for_archiving 1 1
extract_tracks 1 1
forward_periodic_report 1 1
make_routing 1 1
parse_input_file 1 1
prepare_periodic_report 1 1

main oloughln vlo c3i_system_prototype c3i_system.comms_interface.convert_to_text_file

main oloughln vll c3i_system_prototype c3i_system.comms_interface.convert_to_text_file

main oloughln cao c3i_system_prototype c3i_config_1 c3i_system 1 1
main oloughln clo c3i_system_prototype c3i_config_1
c3i_system.comms_interface.convert_to_text_file 1 1
Wed Dec 31 16:00:00 1969
c3i_system.comms_interface.decide_for_archiving 1 1
Wed Dec 31 16:00:00 1969
c3i_system.comms_interface.extract_tracks 1 1
Wed Dec 31 16:00:00 1969
c3i_system.comms_interface.forward_periodic_report 1 1
Wed Dec 31 16:00:00 1969
c3i_system.comms_interface.make_routing 1 1
Wed Dec 31 16:00:00 1969
c3i_system.comms_interface.parse_input_file 1 1
Wed Dec 31 16:00:00 1969
c3i_system.comms_interface.prepare_periodic_report 1 1
Wed Dec 31 16:00:00 1969
c3i_system.comms_interface 1 1
Wed Dec 31 16:00:00 1969
c3i_system.comms_links 1 1
Wed Dec 31 16:00:00 1969
c3i_system.navigation_system 1 1
Wed Dec 31 16:00:00 1969
c3i_system.sensor_interface.analyze_sensor_data 1 1
Wed Dec 31 16:00:00 1969

c3i_system.sensor_interface.prepare_sensor_track 1 1
Wed Dec 31 16:00:00 1969
c3i_system.sensor_interface 1 1
Wed Dec 31 16:00:00 1969
c3i_system.sensors 1 1
Wed Dec 31 16:00:00 1969
c3i_system.track_database_manager.add_comms_track 1 1
Wed Dec 31 16:00:00 1969
c3i_system.track_database_manager.add_sensor_track 1 1
Wed Dec 31 16:00:00 1969
c3i_system.track_database_manager.filter_comms_track 1 1
Wed Dec 31 16:00:00 1969
c3i_system.track_database_manager.filter_sensor_track 1 1
Wed Dec 31 16:00:00 1969
c3i_system.track_database_manager.monitor_ownship_position 1 1
Wed Dec 31 16:00:00 1969
c3i_system.track_database_manager 1 1
Wed Dec 31 16:00:00 1969
c3i_system.user_interface.display_tracks 1 1
Wed Dec 31 16:00:00 1969
c3i_system.user_interface.emergency_status_screen 1 1
Wed Dec 31 16:00:00 1969
c3i_system.user_interface.get_user_inputs 1 1
Wed Dec 31 16:00:00 1969
c3i_system.user_interface.manage_user_interface 1 1
Wed Dec 31 16:00:00 1969
c3i_system.user_interface.message_arrival_panel 1 1
Wed Dec 31 16:00:00 1969
c3i_system.user_interface.message_editor 1 1
Wed Dec 31 16:00:00 1969
c3i_system.user_interface.status_screen 1 1
Wed Dec 31 16:00:00 1969
c3i_system.user_interface 1 1
Wed Dec 31 16:00:00 1969
c3i_system.weapons_interface 1 1
Wed Dec 31 16:00:00 1969
c3i_system.weapons_systems 1 1
Wed Dec 31 16:00:00 1969

main oloughln cll c3i_system_prototype c3i_config_1
comms_interface 1 1

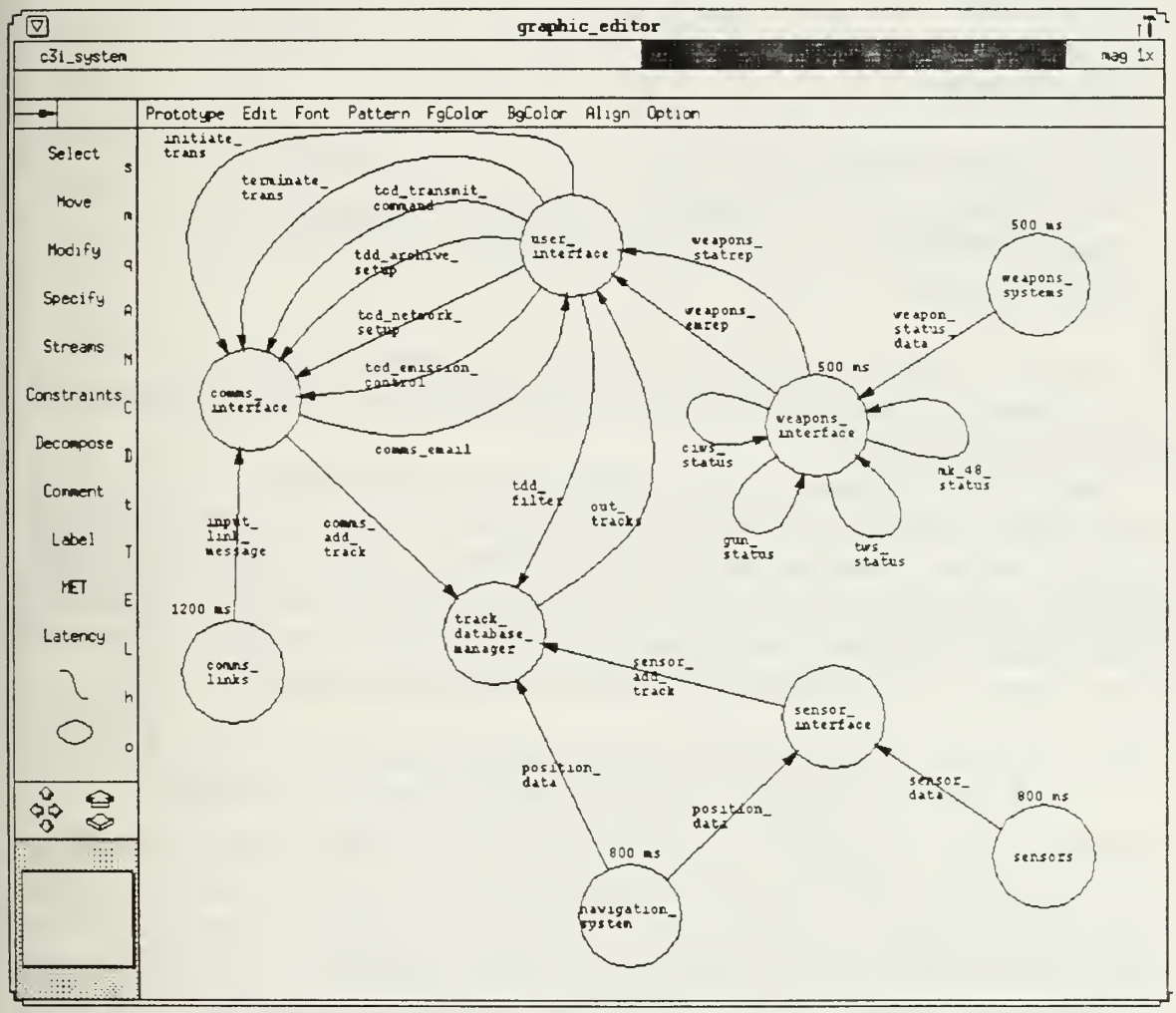
```
comms_links              1 1
navigation_system        1 1
sensor_interface         1 1
sensors                  1 1
track_database_manager   1 1
user_interface           1 1
weapons_interface        1 1
weapons_systems          1 1

main oloughln cdt c3i_system_prototype c3i_config_1 w
c3i_system 11

main oloughln vds c3i_system_prototype c3i_system 1 1
Mon Jun 8 14:51:38 1992
oloughln

main oloughln vdd c3i_system_prototype c3i_system 1 1
Mon Jun 8 14:51:38 1992

main oloughln vgl c3i_system_prototype c3i_system 1 1
Wed Dec 31 16:00:00 1969

main oloughln vgv c3i_system_prototype c3i_system
1 1
main oloughln vgd c3i_system_prototype c3i_system 1 1

main oloughln vud c3i_system_prototype c3i_system descp 1 1
This is a test file added to the design database for example purposes.
Any thing can be added to it, and nothing can be expected back.

main oloughln vgp c3i_system_prototype c3i_system 1 1

main oloughln vgg c3i_system_prototype c3i_system 1 1

main oloughln vgi c3i_system_prototype c3i_system 1 1
 IMPLEMENTATION
 GRAPH
 VERTEX comms_links : 1200 ms
 VERTEX comms_interface
 VERTEX track_database_manager
 VERTEX user_interface
```

VERTEX weapons_interface : 500 ms
VERTEX weapons_systems : 500 ms
VERTEX navigation_system : 800 ms
VERTEX sensor_interface
VERTEX sensors : 800 ms
EDGE input_link_message comms_links -> comms_interface
EDGE tcd_network_setup user_interface -> comms_interface
EDGE tdd_archive_setup user_interface -> comms_interface
EDGE tcd_transmit_command user_interface -> comms_interface
EDGE terminate_trans user_interface -> comms_interface
EDGE tcd_emission_control user_interface -> comms_interface
EDGE initiate_trans user_interface -> comms_interface
EDGE comms_add_track comms_interface -> track_database_manager
EDGE comms_email comms_interface -> user_interface
EDGE position_data navigation_system -> track_database_manager
EDGE sensor_add_track sensor_interface -> track_database_manager
EDGE tdd_filter user_interface -> track_database_manager
EDGE out_tracks track_database_manager -> user_interface
EDGE weapons_emrep weapons_interface -> user_interface
EDGE weapons_statrep weapons_interface -> user_interface
EDGE weapon_status_data weapons_systems -> weapons_interface
EDGE position_data navigation_system -> sensor_interface
EDGE sensor_data sensors -> sensor_interface
DATA STREAM
initiate_trans : initiate_transmission_sequence,
terminate_trans : BOOLEAN,
comms_add_track : add_track_tuple,
position_data : ownship_navigation_info,
position_data : ownship_navigation_info,
sensor_data : sensor_record,
sensor_add_track : add_track_tuple,
tdd_filter : set_track_filter,
out_tracks : track_tuple,
weapons_statrep : weapon_status_report,
weapons_emrep : weapon_status_report,
weapon_status_data : weapon_status,
comms_email : filename,
tcd_transmit_command : transmit_command,
tdd_archive_setup : archive_setup,
tcd_network_setup : network_setup,
tcd_emission_control : emissions_control_command,

```
input_link_message : filename

CONTROL CONSTRAINTS
OPERATOR comms_links
PERIOD 50 SEC

OPERATOR comms_interface

OPERATOR user_interface

OPERATOR track_database_manager

OPERATOR navigation_system
PERIOD 50 SEC

OPERATOR sensor_interface

OPERATOR sensors
PERIOD 50 SEC

OPERATOR weapons_interface
TRIGGERED BY SOME
weapon_status_data
 IF weapon_status_data.status = DAM OR weapon_status_data.status =
service_required OR weapon_status_data.status = out_of_ammunition

OPERATOR weapons_systems
PERIOD 50 SEC
END

main oloughln vgc c3i_system_prototype c3i_system 1 1
OPERATOR c3i_system
SPECIFICATION
DESCRIPTION { UNDEFINED_TEXT }
END

END COMMAND LINE TEST
```

B. EXAMPLES WITH CAPS SCREEN DUMPS.

The testing conducted in section A was executed with a shell script. In this section a certain selection of design database commands will be executed to show how variation and versioning work. The examples are accompanied with screen dumps of CAPS graphic editor screens. These screens will display the changes made using the graphic editor which will correspond to the variations and versions stored in the database.



SCREEN 1. c3i_system Variation 1, Version 1

Screen 1 shows a top level view of c3i_system. This view is stored in the database as object c3i_system variation 1, version 1. To access this object the operator name on the command line would be c3i_system. The actual name in the database is c3i_system_1, a concatenation of the object name and the variation number. The designers should not concern themselves the with the internal database name. The DDBMS handles the concatenation of the object name and variation number and also stripping off the variation number when the object name needs to be retrieved.

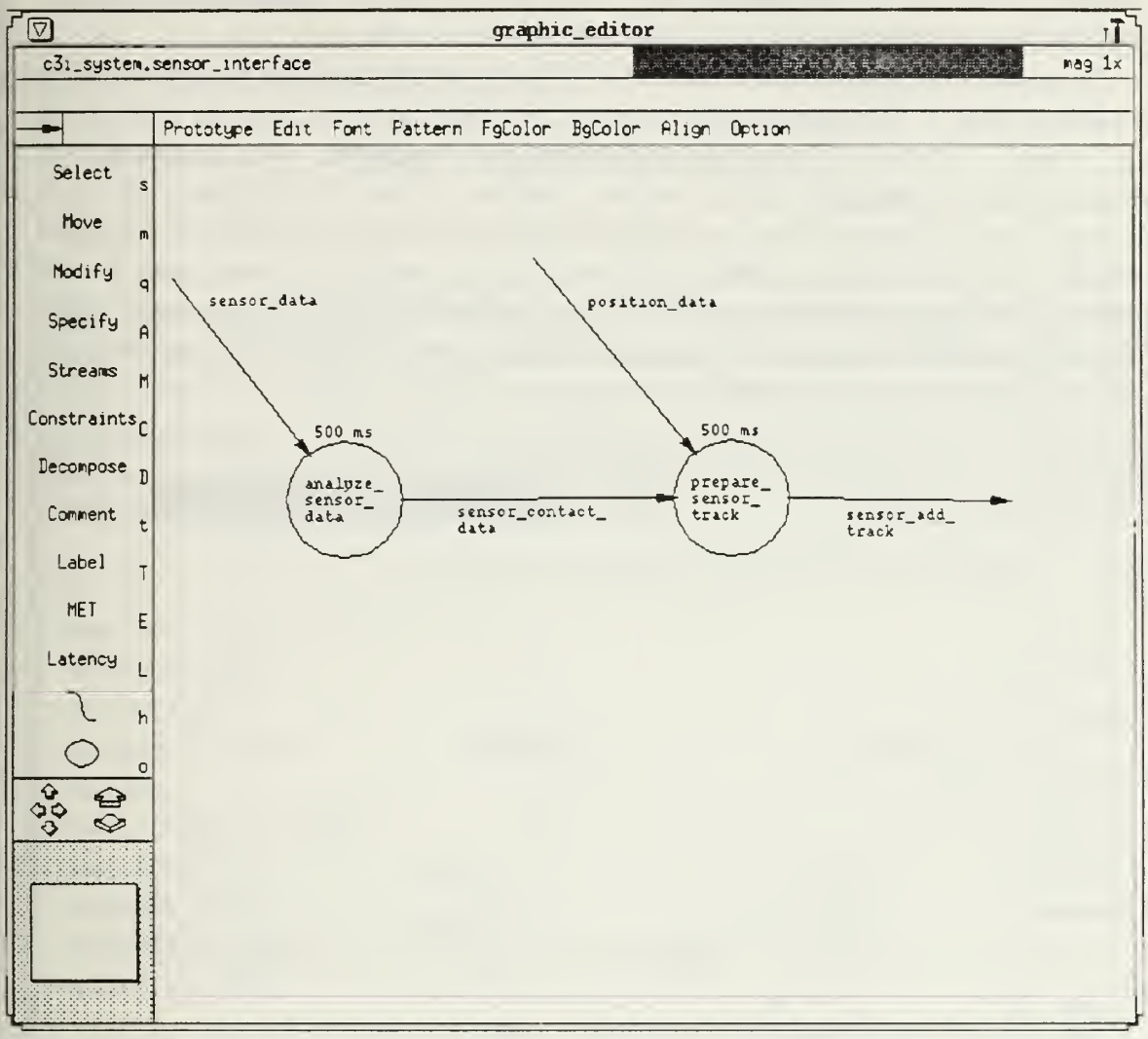
The command issued below retrieves from the database the object c3i_system variation 1, version 1, and displays all the operators and their variation and version numbers which are children of c3i_system. The listing, like the graphic editor view only displays the current level view.

```
main oloughln vll c3i_system_prototype c3i_system 1 1
comms_interface          1 1
comms_links              1 1
navigation_system        1 1
sensor_interface         1 1
sensors                  1 1
track_database_manager   1 1
user_interface           1 1
weapons_interface        1 1
weapons_systems          1 1
```

One name is listed for each object displayed in the graphic editor.

c3i_system is a composite object which can be decomposed on the command line as well as in the graphic editor.

Screen 2, is a graphic editor display of the decomposition of the object sensor_interface. The listing shows that two objects make up c3i_system.sensor_interface variation 1, version 1. These two objects are both variation 1, version 1. These two displays demonstrate composite and atomic objects. c3i_system is a composite object composed of the nine objects listed above and displayed on screen 1. sensor_interface is a composite object composed of the two objects listed below and displayed on screen 2.



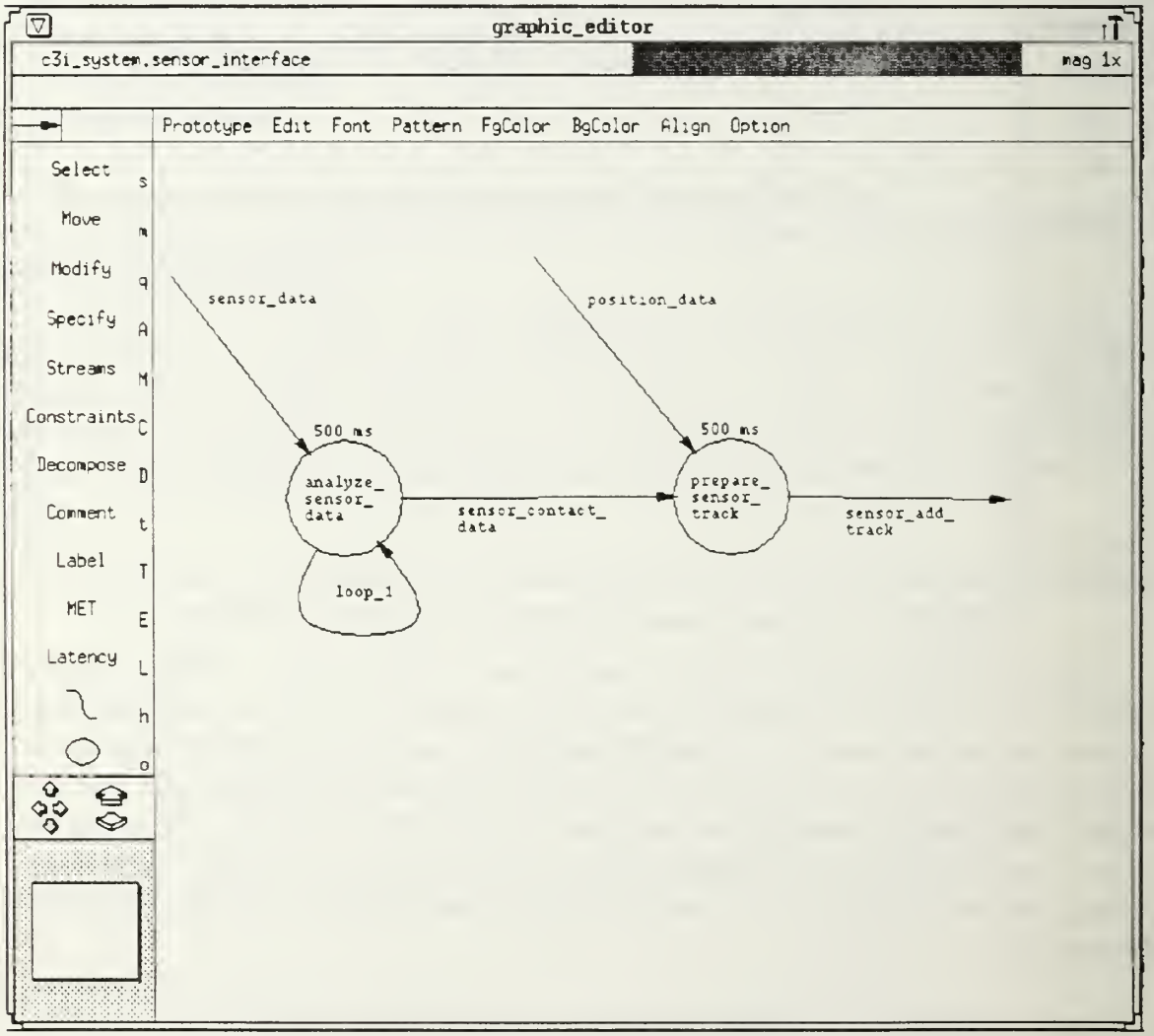
SCREEN 2. c3i_system.sensor_interface, Variation 1, Version 1

The two objects displayed above make up the composite object, sensor_interface variation 1, version 1. The command below gives a listing of the objects names and their variation and version numbers. As with the view of c3i_system, the graphic display shows only the current level. The same is true of the object listed on the command line. The two objects listed from the command line are atomic objects. This is verified by executing the vll command on either of the two objects listed. As shown below, no output is sent to the screen. If no output is sent to the screen this signifies that the object is atomic and will not decompose.

```
main oloughln vll c3i_system_prototype c3i_system.sensor_interface 1 1
analyze_sensor_data                1 1
prepare_sensor_track               1 1
```

main oloughln vll c3i_system_prototype c3i_system.sensor_interface.prepare_sensor_track 1 1

On screen 3 the graphic editor shows that prepare_sensor_track has been modified by the addition loop_1. The graphic editor displays the object retrieved from the database. In this case it is c3i_system.sensor_interface variation 1, version 1. c3i_system.sensor_interface will not version, because of this modification, until it is reinserted into the database. After the modified object has been reinserted into the database the object c3i_system.sensor_interface will version and so will all its parents, until the root object has versioned. In this example c3i_system.sensor_interface.analyze_sensor_data, c3i_system.sensor_interface and c3i_system will all version. The following command from the command line demonstrates this versioning.



SCREEN 3. c3i_system.sensor_interface, Variation 1, Version 2

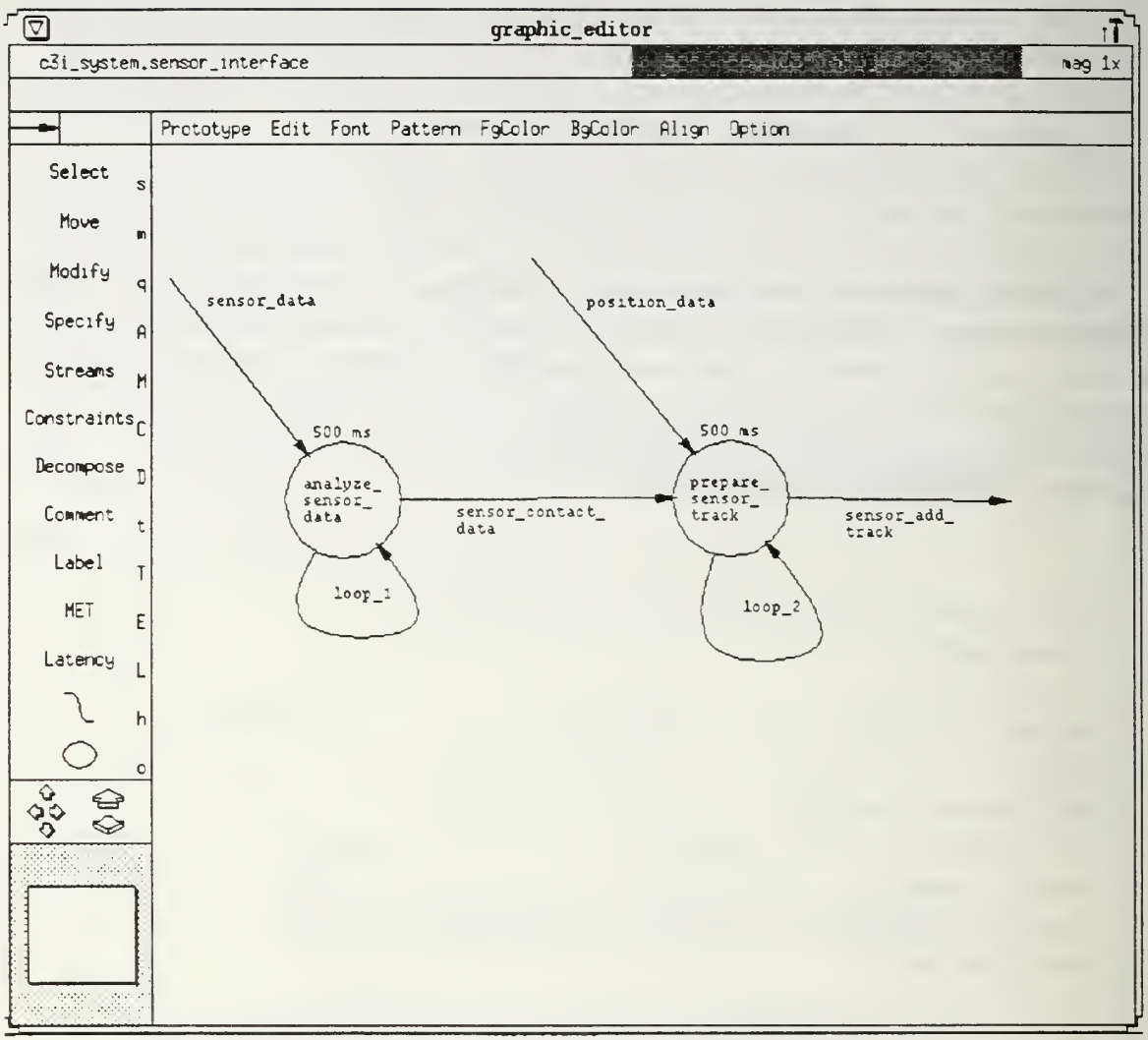
The vgv command lists all the variations and versions in the database for the c3i_system. As shown there are now two c3i_systems, c3i_systen variation 1, version 1 and c3i_systen variation 1, version 2.

```
main oloughln vgv c3i_system_prototype c3i_system
1 1
1 2
```

The vll command now shows that sensor_interface has versioned also. So c3i_system variation1, version 2 contain sensor_interface variation 1, version 2. While c3i_system variation1, version1 contains object sensor_interface variation 1, version 1. Further decomposition shows c3i_system.sensor_interface variation 1, version 2 is made up of prepare_sensor_track variation1, version 1 and analyze_sensor_data variation 1, version 2. This reflects the changes made in the graphic editor.

```
main oloughln vll c3i_system_prototype c3i_system 1 2
comms_interface                 1 1
comms_links                     1 1
navigation_system               1 1
sensors                         1 1
track_database_manager          1 1
user_interface                  1 1
weapons_interface               1 1
weapons_systems                 1 1
sensor_interface                1 2

main oloughln vll c3i_system_prototype c3i_system.sensor_interface 1 2
prepare_sensor_track            1 1
analyze_sensor_data             1 2
```

Screen 4 shows a modification to the atomic object prepare_sensor_track variation1, version 1. When reinserted into the database, this object and its parents up to and including the root object will version. This is demonstrated with the following commands.



SCREEN 4. c3i_system.sensor_interface, Variation 1, Version 3

main oloughln vgv c3i_system_prototype c3i_system
1 1
1 2
1 3

There are now 3 version of variation 1 of object c3i_system in the database. c3i_system variation 1, version 3 contains the following objects.

main oloughln vll c3i_system_prototype c3i_system 1 3

comms_interface	1 1
comms_links	1 1
navigation_system	1 1

```
sensors                    1 1
track_database_manager     1 1
user_interface             1 1
weapons_interface          1 1
weapons_systems            1 1
sensor_interface           1 3
```

The object analyze_sensor_data variation 1, version 2 was not effected by the versioning of prepare_sensor_track variation 1, version 1 because it is not in the direct path between prepare_sensor_track variation 1, version 1 and the root object. In this example c3i_system.sensor_interface.prepare_sensor_data, c3i_system.sensor_interface and c3i_system all version.

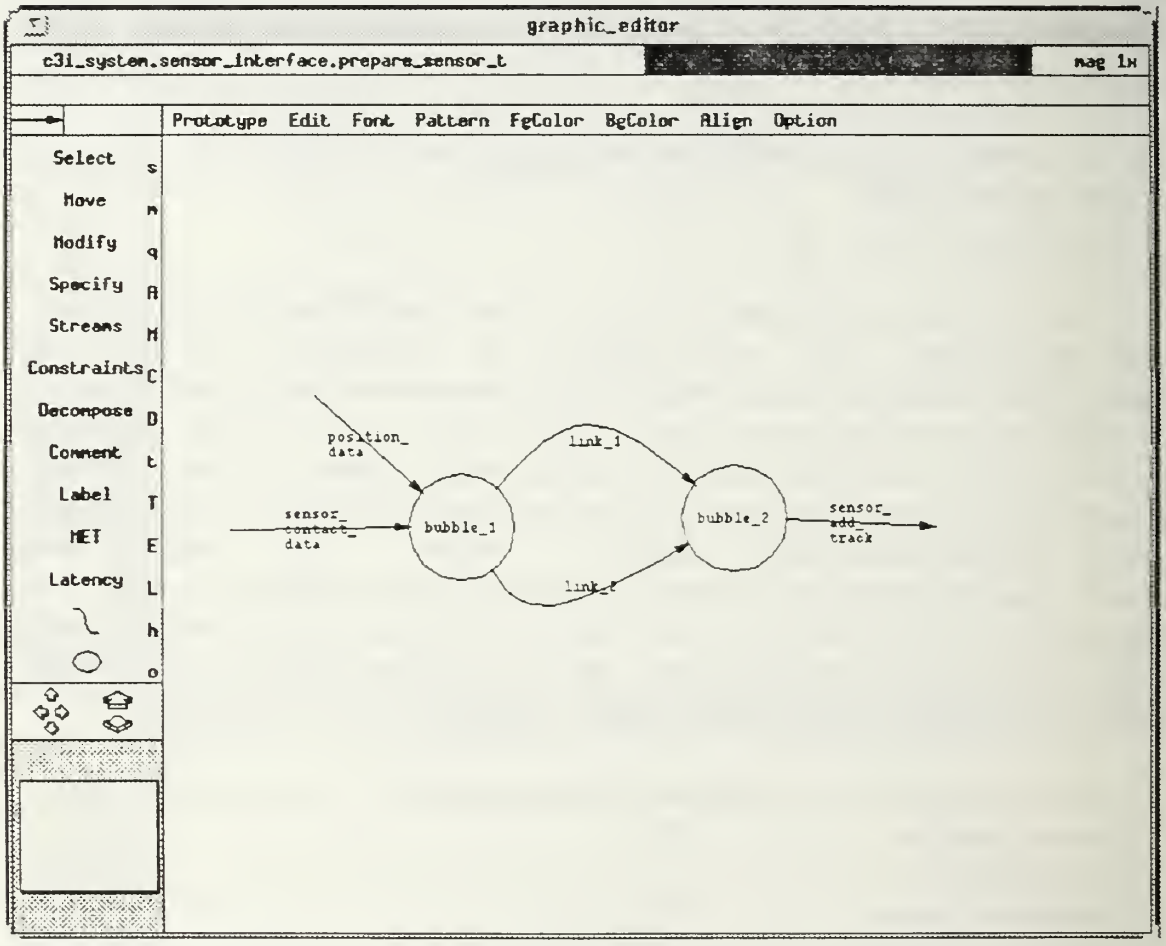
```
main oloughln vll c3i_system_prototype c3i_system.sensor_interface 1 3
analyze_sensor_data        1 2
prepare_sensor_track       1 2
```

In this next example c3i_system variation 1, version 2 was retrieved from the database. Using the graphic editor sensor_interface was decomposed and then the prepare_sensor_track object was decomposed. Upon reinsertion of prepare_sensor_track a new object was created and prepare_sensor_track became a composite object containing the new objects constructed with the graphic editor. Screen 5 shows the new prepare_sensor_track object created. As before each parent of the new object versioned. Because prepare_sensor_track variation 1, version 1 was modified and prepare_sensor_track variation 1, version 2 already exists a new variation was created. This happened to each parent because a parent object with a greater version number already existed. The following commands show the new variations created because of the modification of prepare_sensor_track variation 1, version 1.

```
main oloughln vll c3i_system_prototype c3i_system 2 1
comms_interface            1 1
comms_links                1 1
navigation_system          1 1
sensors                    1 1
track_database_manager     1 1
user_interface             1 1
weapons_interface          1 1
weapons_systems            1 1
sensor_interface           2 1
```

main oloughln vll c3i_system_prototype c3i_system.sensor_interface 2 1
analyze_sensor_data 1 2
prepare_sensor_track 2 1

main oloughln vll c3i_system_prototype c3i_system.sensor_interface.prepare_sensor_track 2 1
bubble_1 1 1
bubble_2 1 1



SCREEN 5. c3i_system.sensor_interface,.prepare_sensor_track Variation 2, Version 1

The object prepare_sensor_track variation 2, version 1 contains the objects bubble_1 and bubble_2, both of which are variation 1, version 1. In this example a new level is created causing the parent objects to version and create new versions, alternate paths of development, because versions of a higher number existed on the same variation.

C. TEST SHELL SCRIPT.

```
#!/usr/local/bin/tcsh
echo "START COMMAND LINE TEST "
echo
echo
echo "main oloughln pip c3i_system_prototype 'Michael D. O'Loughlin' descp "
main oloughln pip c3i_system_prototype "Michael D. O'Loughlin" descp
echo
echo
echo "main oloughln pln "
main oloughln pln
echo
echo
echo "main oloughln pll "
main oloughln pll
echo
echo
echo "main oloughln pgd c3i_system_prototype "
main oloughln pgd c3i_system_prototype
echo
echo
echo "main oloughln pud c3i_system func_description "
main oloughln pud c3i_system_prototype func_description
echo
echo
echo "main oloughln pgd c3i_system_prototype "
main oloughln pgd c3i_system_prototype
echo
echo
echo "main oloughln prd c3i_system_prototype "
main oloughln prd c3i_system_prototype
echo
echo
echo "main oloughln pgl c3i_system_prototype "
main oloughln pgl c3i_system_prototype
echo
echo
echo "main oloughln pul c3i_system_prototype 'Larry Williamson' "
main oloughln pul c3i_system_prototype "Larry Williamson"
echo
```

```
echo
echo "main oloughln pgl c3i_system_prototype "
main oloughln pgl c3i_system_prototype
echo
echo
echo "main oloughln pds c3i_system_prototype"
main oloughln pds c3i_system_prototype
echo
echo
#-x-x-x
```

```
echo
echo
echo "main oloughln cud c3i_system_prototype c3i_config_1 func_description "
main oloughln cud c3i_system_prototype c3i_config_1 func_description
echo
echo
echo "main oloughln cgd c3i_system_prototype c3i_config_1 "
main oloughln cgd c3i_system_prototype c3i_config_1
echo
echo
echo "main oloughln cpl c3i_system_prototype c3i_config_1 'This is the first Post to c3i_system Log' "
main oloughln cpl c3i_system_prototype c3i_config_1 "This is the first Post to c3i_system Log"
echo
echo
echo "main oloughln cgl c3i_system_prototype c3i_config_1 "
main oloughln cgl c3i_system_prototype c3i_config_1
echo
echo
echo "main oloughln cpl c3i_system_prototype c3i_config_1 'This is the second Post to c3i_config_1 Log' "
main oloughln cpl c3i_system_prototype c3i_config_1 "This is the second Post to c3i_config_1 Log"
echo
echo
echo "main oloughln cgl c3i_system_prototype c3i_config_1 "
main oloughln cgl c3i_system_prototype c3i_config_1
echo
echo
echo "main oloughln cln c3i_system_prototype"
main oloughln cln c3i_system_prototype
echo
echo
echo "main oloughln cds c3i_system_prototype c3i_config_1 "
main oloughln cds c3i_system_prototype c3i_config_1
echo
echo
#-x-x-x
```

```
echo "main oloughln vaa c3i_system_prototype c3i_system "
main oloughln vaa c3i_system_prototype c3i_system
echo
echo
echo "main oloughln pll "
main oloughln pll
echo
echo
echo "main oloughln vlo c3i_system_prototype c3i_system "
main oloughln vlo c3i_system_prototype c3i_system
echo
echo
echo "main oloughln vll c3i_system_prototype c3i_system "
main oloughln vll c3i_system_prototype c3i_system
echo
echo
echo "main oloughln vlo c3i_system_prototype c3i_system.comms_interface "
main oloughln vlo c3i_system_prototype c3i_system.comms_interface
echo
echo
echo "main oloughln vll c3i_system_prototype c3i_system.comms_interface "
main oloughln vll c3i_system_prototype c3i_system.comms_interface
echo
echo
echo "main oloughln vlo c3i_system_prototype
c3i_system.comms_interface.convert_to_text_file "
main oloughln vlo c3i_system_prototype
c3i_system.comms_interface.convert_to_text_file
echo
echo
echo "main oloughln vll c3i_system_prototype
c3i_system.comms_interface.convert_to_text_file "
main oloughln vll c3i_system_prototype
c3i_system.comms_interface.convert_to_text_file
echo
echo
echo "main oloughln cao c3i_system_prototype c3i_config_1 c3i_system 1 1 "
main oloughln cao c3i_system_prototype c3i_config_1 c3i_system 1 1
echo
echo
echo "main oloughln clo c3i_system_prototype c3i_config_1 "
```

```
    main oloughln clo c3i_system_prototype c3i_config_1
    echo
    echo
    echo "main oloughln cll c3i_system_prototype c3i_config_1 "
    main oloughln cll c3i_system_prototype c3i_config_1
    echo
    echo
    echo "main oloughln cdt c3i_system_prototype c3i_config_1 w "
    main oloughln cdt c3i_system_prototype c3i_config_1 w
    echo
    echo
    #-x-x-x
```

```
echo
echo "main oloughln vgg c3i_system_prototype c3i_system 1 1 "
#main oloughln vgg c3i_system_prototype c3i_system 1 1
echo
echo
echo "main oloughln vgi c3i_system_prototype c3i_system 1 1 "
main oloughln vgi c3i_system_prototype c3i_system 1 1
echo
echo
echo "main oloughln vgc c3i_system_prototype c3i_system 1 1 "
main oloughln vgc c3i_system_prototype c3i_system 1 1
echo
echo
echo "main oloughln caa c3i_system_prototype c3i_config_1 "
main oloughln cdt c3i_system_prototype c3i_config_1
echo
echo
echo "END COMMAND LINE TEST "
```

APPENDIX D

A. SOURCE CODE.

```
// File Header ---------------------------------------------------------
//
// Filename     : main.cxx
// SCCS ID      : 1.3
// Release No.  : 1
// Date         : 9/16/91
// Author       : Garry Lewis
//              : Drew Dwyer
//.Modified by...: Michael D. O'Loughlin
//.Modifications.: The modifications made to this module correspond to the
//              modicications made to ....., as of .....
//              The modifications made to this module will, .....
//              The following operations were added or modified:
//                1.
//                2.
//                3.
//                4.
//              The above modifications were made on .....

// Compiler     : Glockenspiel C++ 2.1
//
// End header comments -------------------------------------------------

// SCCS ID follows: will compile to place date/time stamp in
// object file

static char main_Sccsld[] = "@(#)main.cxx       1.3\t9/16/91";

// Interface Dependencies ----------------------------------------------

//
#define DEBUG
#include "debug.h"

#ifndef __DDBDEFINES_H
#include "ddbdefines.h"
#endif

#include "My_String.h"

#include <Object.h>
#include <Transaction.h>
#include <Directory.h>
#include <GlobalEntities.h>
```

```
#include <Database.h>
#include <List.h>
#include <stream.hxx>

extern "C--"
{
#include <stdlib.h>
#include <stddef.h>
#include <string.h>
#include <ctype.h>
}

#ifndef __VOBJECTFUNC_H
#include "vobjectfunc.h"
#endif

#ifndef __EVALUATION_H
#include "evaluation.h"
#endif

#ifndef __PROTFUNC_H
#include "protfunc.h"
#endif

#ifndef __CONFFUNC_H
#include "conffunc.h"
#endif

// End Interface Dependencies ----------------------------------------------

Type *V_OBJECT_OType;
Type *THREAD_OType;
Type *COMPONENT_OType;
Type *TEXT_OBJECT_OType;
Type *PROTOTYPE_OType;
Type *CONFIGURATION_OType;
Type *DDB_OType;

Directory *prototype_dir = (Directory*)0;
Directory *ddbRootDir = (Directory*)0;
List *myPrototypeList = (List*)0;
char *database_name = (char *)0;
char *userPtr = (char *)0;
char *dirNamePtr = (char *)0;
```

```
void initialize_types(void);
void setUpDirectory(char *, char *);

int main(int argc, char *argv[])

{
 FTRACER("main", "DDBCAPS.log", 1);
 dirNamePtr = getenv("PROTOTYPE");
 userPtr = getenv("USER");
 Boolean done=FALSE;
 int menu_option=0;

 ifstream inFile;

 char *function_tag;

 int number_arguments;
 int run = 0;

 if (argv[1])
  {
   database_name = new char[strlen(argv[1])+1];
   strcpy(database_name,argv[1]);
  }

 if (argv[2])
  {
   function_tag = new char[strlen(argv[2])+1];
   strcpy(function_tag, argv[2]);
  }

 number_arguments = argc - 3;
 // subtract the "design" "database" & "function"
 // from the argument(s) we evaluate.

 if (number_arguments < 0)
  {
   cerr << "<ERROR: Not enough arguments specified>\n\n"
    << "   Format for Usage: \n\n"
     << "   designdb dbasename function [argument, ...]\n\n";
   // INSERT CODE TO CLOSE TRACER FILE
   SETLMODE;
   DESTROYTRACER;
   exit(1);
```

```
  }

if (OC_open(database_name))
  {
   OC_transactionStart();

   initialize_types();

   setUpDirectory(argv[3], function_tag);
  }
else
  {
   cerr << "<ERROR: Error attempting to open database "
    << database_name
     << " ---> database does not exist in registry\n\n\n";
   // INSERT CODE TO CLOSE TRACER FILE
   SETLMODE;
   DESTROYTRACER;
   exit(1);
  }

if (strlen(function_tag)<3 || strlen(function_tag)>3)
  {
   cerr << "<ERROR: illegal function type--> <"
    << function_tag << "> >\n";
  }
else
  {
   if (function_tag[0] == 'P' || function_tag[0]=='p')
    run = evaluate_prototype_function(upper(function_tag),
                                      number_arguments);
   else if (function_tag[0] == 'C' || function_tag[0]=='c')
    run=evaluate_configuration_function(upper(function_tag),
                                        number_arguments);
   else if (function_tag[0] == 'V' || function_tag[0]=='v')
    run = evaluate_vobject_function(upper(function_tag),
                                    number_arguments);
   else if (function_tag[0] == 'H' || function_tag[0]=='h')
    run = evaluate_historical_function(upper(function_tag),
                                    number_arguments);
   else
    {
     cerr << "<ERROR: illegal function type--> <";
     cerr << function_tag << "> >\n";
```

```
    }
  }


switch (run)
  {
  case LIST_PROTOTYPES:
    list_prot_func(number_arguments);
    break;
  case LONG_LIST_PROTOTYPES:
    long_list_prot_func(number_arguments);
    break;
  case GET_PROTOTYPE_LEADER:
    get_prot_leader_func(number_arguments, argv[3]);
    break;
  case DUMP_PROTOTYPE_SUMMARY:
    dump_prot_summary_func(number_arguments, argv[3]);
    break;
  case GET_PROTOTYPE_DESCRIPTION:
    get_prot_description_func(number_arguments, argv[3]);
    break;
  case RETRIEVE_PROTOTYPE_DATE:
    retrieve_prot_date_func(number_arguments, argv[3]);
    break;
  case INSERT_PROTOTYPE:
    insert_prot_func(number_arguments, argv[3],
                argv[4], argv[5]);
    break;
  case UPDATE_PROTOTYPE_LEADER:
    update_prot_leader_func(number_arguments,
                    argv[3], argv[4]);
    break;
  case UPDATE_PROTOTYPE_DESC:
    update_prot_desc_func(number_arguments,
                    argv[3], argv[4]);
    break;
  case UPDATE_PROTOTYPE_NAME:
    update_prot_name_func(number_arguments,
                    argv[3], argv[4]);
    break;

  case PROTOTYPE_VARIATION_LIST:
    list_prot_variations(number_arguments);
    break;
```

```
case PROTOTYPE_VERSION_LIST:
  list_prot_versions(number_arguments);
  break;
case PROTOTYPE_VAR_AND_VER_LIST:
  list_prot_var_and_ver(number_arguments);
  break;

case DUMP_CONFIGURATION_SUMMARY:
  dump_conf_summary_func(number_arguments,
                argv[3],argv[4]);
  break;
case GET_LATEST_CONFIGURATION:
  get_latest_conf_func(number_arguments,
                argv[3]);
  break;
case ADD_CONFIGURATION_OPERATORS:
  add_conf_operators_func(number_arguments,
                argv[3],argv[4]);
  break;
case DUMP_CONFIGURATION_OPERATORS:
  dump_conf_operators_func(number_arguments,
                argv[3],argv[4],argv[5]);
  break;
case RELEASE_CONFIGURATION_LOCK:
  release_conf_lock_func(number_arguments,
                argv[3],argv[4]);
  break;
case LONG_LIST_CONFIGURATION_OPERATORS:
  long_list_conf_operators_func(number_arguments,
                argv[3],argv[4]);
  break;
case LIST_CONFIGURATION_OPERATORS:
  list_conf_operators_func(number_arguments,
                argv[3],argv[4]);
  break;
case LIST_CONFIGURATION_DEFAULT_OPERATOR:
  list_conf_default_operator_func(number_arguments,
                argv[3],argv[4]);
  break;
case LIST_CONFIGURATIONS:
  list_conf_func(number_arguments, argv[3]);
  break;
case UPDATE_CONFIGURATION_NAME:
  update_conf_name_func(number_arguments,
```

```
                argv[3], argv[4],
                argv[5]);
  break;
case GET_CONFIGURATION_DESCRIPTION:
  get_conf_desc_func(number_arguments,
              argv[3], argv[4]);
  break;
case INSERT_CONFIGURATION:
  insert_conf_func(number_arguments, argv[3],
            argv[4], argv[5], argv[6]);
  break;
case UPDATE_CONFIGURATION_DESCRIPTION:
  update_conf_desc_func(argv[3], argv[4],
                argv[5]);
  break;
case GET_CONFIGURATION_MANAGER:
  get_conf_manager_func(number_arguments,
                argv[3], argv[4]);
  break;
case UPDATE_CONFIGURATION_MANAGER:
  update_conf_manager_func(number_arguments, argv[3],
                  argv[4], argv[5]);
  break;
case GET_CONFIGURATION_DATE:
  get_conf_date_func(number_arguments,
              argv[3], argv[4]);
  break;
case GET_CONFIGURATION_CHANGED:

case POST_CONFIGURATION_LOG:
  post_conf_log_func(argv[3], argv[4], argv[5]);
  break;
case GET_CONFIGURATION_LOG:
  get_conf_log_func(number_arguments,
              argv[3], argv[4]);
  break;
case ATTACH_OPERATOR:
  attach_vobject_to_conf_func(number_arguments,
                    argv[3],argv[4],
                    argv[5],argv[6],argv[7]);
  break;
  // ------X----------X----------X----------X
  //
  //  VOBJECT CASE STATEMENTS
```

```
 //
 // ------X-----------------X----------------X-----------------X
case LIST_OPERATORS:
 list_operators_func(number_arguments,argv[3],
              argv[4],argv[5],argv[6]);
 break;
case GET_VOBJECT_DESCRIPTION:
 get_vobject_desc_func(number_arguments, argv[3],
               argv[4], argv[5], argv[6]);
 break;
case UPDATE_VOBJECT_DESCRIPTION:
 update_vobject_desc_func(number_arguments,
                 argv[3], argv[4],
                 argv[5], argv[6], argv[7]);
 break;
case GET_VOBJECT_DATE:
 get_vobject_date_func(number_arguments, argv[3],
               argv[4], argv[5], argv[6]);
 break;
case GET_VOBJECT_VERSIONS:
 get_vobject_versions_func(number_arguments,
                 argv[3], argv[4]);
 break;
case GET_VOBJECT_LOCK:
 get_vobject_lock_func(number_arguments,
               argv[3], argv[4],
               argv[5], argv[6]);
 break;
case GET_VOBJECT_VERSION:
 get_vobject_version_func();
 break;

case DUMP_VOBJECT_SUMMARY:
 dump_vobject_summary_func(number_arguments, argv[3],
                 argv[4], argv[5], argv[6]);
 break;
case GET_VOBJECT_POSTSCRIPT:
 get_vobject_psfile_func(number_arguments, argv[3],
                argv[4], argv[5],argv[6]);
 break;
case GET_VOBJECT_GRAPH:
 get_vobject_graphfile_func(number_arguments,argv[3],
                 argv[4], argv[5],argv[6]);
 break;
```

```
case GET_VOBJECT_IMPLEMENTATION:
 get_vobject_impfile_func(number_arguments, argv[3],
                argv[4], argv[5],argv[6]);
 break;
case GET_VOBJECT_SPECIFICATION:
 get_vobject_specfile_func(number_arguments, argv[3],
                argv[4], argv[5],argv[6]);
 break;
case GET_VOBJECT_SOURCE:
 get_vobject_sourcefile_func(number_arguments,
                argv[3], argv[4],
                argv[5],argv[6]);
 break;
case DUMP_VOBJECT_FILES:
 dump_vobject_files_func(number_arguments,
                argv[3], argv[4],
                argv[5],argv[6], argv[7]);
 break;
case DUMP_VOBJECT_TREE:
 dump_vobject_tree_func(number_arguments, argv[3],
                argv[+4], argv[5], argv[6], argv[7]);
 break;

case ADD_NEW_VARIATION:
 add_new_variation_func(number_arguments,
                argv[3],argv[4],
                argv[5],argv[6]);
 break;

case RELEASE_OPERATOR_LOCK:
 release_operator_lock_func(number_arguments,
                argv[3],argv[4],
                argv[5],argv[6]);
 break;
case RELEASE_SUBTREE_LOCK:
 release_subtree_lock_func(number_arguments,
                argv[3],argv[4],
                argv[5],argv[6]);
 break;
case LONG_LIST_OPERATORS:
 long_list_operators_func(number_arguments,argv[3],
                argv[4],argv[5],argv[6]);
 break;
case LONG_LIST_CHILDREN:
```

```
    long_list_children_func(number_arguments,argv[3],
                    argv[4],argv[5]);
    break;
   case LONG_LIST_PARENTS:
    long_list_parents_func(number_arguments,argv[3],
                    argv[4],argv[5], argv[6]);
    break;
   case ERROR_IN_EVALUATION:
    cerr << "<ERROR: Error returned from evaluation"
     << " function>\n";
    break;

   case HISTORICAL_TRAIL:
    historical_trail(number_arguments,argv[3],
                argv[4],argv[5], argv[6]);
    break;

   default:
    {
     cerr << "<ERROR: Unknown error with function "
      << " call..switch (run)...>\n";
     // INSERT CODE TO CLOSE TRACER FILE
     SETLMODE;
     SETLMODE;
     DESTROYTRACER;
     exit(1);
    }
   }
 OC_transactionCommit();
 OC_close(database_name);
 /* 1-27-92 cerr << "\n\nDesign Database              "
  << "v1.1 Copyright 1991(c)  WP.D.A.L.G Inc.\n\n\n";  */
 // INSERT CODE TO CLOSE TRACER FILE
 DESTROYTRACER;
 exit(0);
}

// ------------------------------------------------------
// END OF MAIN
//
// ------------------------------------------------------

void initialize_types(void)
{
```

```
 THREAD_OType=(Type *)OC_lookup("THREAD");
 COMPONENT_OType=(Type *)OC_lookup("COMPONENT");
 V_OBJECT_OType=(Type *)OC_lookup("V_OBJECT");
 TEXT_OBJECT_OType=(Type *)OC_lookup("TEXT_OBJECT");
 PROTOTYPE_OType=(Type *)OC_lookup("PROTOTYPE");
 CONFIGURATION_OType = (Type *)OC_lookup("CONFIGURATION");
 DDB_OType = (Type *)OC_lookup("DDB");
};

void setUpDirectory(char *protName, char *func_tag)
{

 ddbRootDir = (Directory *) OC_lookup(DESIGN_DATABASE_DIRECTORY);
 if(ddbRootDir)
   {
    OC_setWorkingDirectory(ddbRootDir);
    myPrototypeList = (List *) OC_lookup(PROTOTYPE_LIST);
   }
 else
   {
    ddbRootDir =  new Directory(DESIGN_DATABASE_DIRECTORY);
    ddbRootDir -> putObject();
    OC_setWorkingDirectory(ddbRootDir);
    myPrototypeList = new List(OC_string,PROTOTYPE_LIST);
    myPrototypeList -> putObject();
   }
 char *yourDirChoice;
 if ((strcmp(func_tag, LIST_PROTOTYPE_UPC) == 0) ||
    (strcmp(func_tag, LIST_PROTOTYPE_LC) == 0) ||
    (strcmp(func_tag, LONG_LIST_PROTOTYPE_LC) == 0) ||
    (strcmp(func_tag, LONG_LIST_PROTOTYPE_UPC) == 0) ||
    (strcmp(func_tag, PROTOTYPE_LIST_VARIATION_LC) == 0) ||
    (strcmp(func_tag, PROTOTYPE_LIST_VARIATION_UPC) == 0) ||
    (strcmp(func_tag, PROTOTYPE_LIST_VERSION_LC) == 0) ||
    (strcmp(func_tag, PROTOTYPE_LIST_VERSION_UPC) == 0) ||
    (strcmp(func_tag, PROTOTYPE_LIST_ALL_LC) == 0) ||
    (strcmp(func_tag, PROTOTYPE_LIST_ALL_UPC) == 0))

   {
    return;
   }
 else
   {
```

```
    yourDirChoice = (char*)(My_String(protName) + My_String("_dir"));
  }
 prototype_dir = (Directory *)OC_lookup(yourDirChoice);

 if ((!(strcmp(func_tag,INSERT_PROTOTYPE_UPC))==0) &&
   (!(strcmp(func_tag,INSERT_PROTOTYPE_LC))==0) &&
   (!(prototype_dir)))
  {
   cerr << "<ERROR: Prototype " << protName
    << " not found>\n"
     << "<Did you remember to run "
      << "INSERT PROTOTYPE first?>\n";
   OC_transactionCommit();
   OC_close(database_name);
// INSERT CODE TO CLOSE TRACER FILE
   exit(0);
  }
 if (!prototype_dir)
  {
   prototype_dir = new Directory(yourDirChoice);
   prototype_dir->putObject();
   if (!prototype_dir)
    cerr << "Did not setup database directory\n";
   OC_setWorkingDirectory(prototype_dir);
  }
 else
  {
   OC_setWorkingDirectory(prototype_dir);
  }

}
```

```
// File Header ---------------------------------------------------------
//.............:
//.Filename.....: component.h
// Date        : 9/16/91
// Author      : Garry Lewis
//             : Drew Dwyer
// Modified by  : Michael D. O'Loughlin
// Date        : 6/18/92
// Modifications : Extensive modifications have been conducted on almost all of the
/*              original code written by Dwyer and Lewis.  Every function that
                accesses an attribute of an object or the entire object has had
                to be modified. The original code is not even recognizable in
                some functions.
                Because the functionality of the design database system changed
                completely from what Dwyer and Lewis developed each individual
                modification has not been documented. Because of the massive
                changes required I discarded a lot of obsolete code and build a
                new system on the base Dwyer and Lewis had established.
*/
// Compiler     : Glockenspiel C++ 2.1
//
// End header comments -----------------------------------------------



#ifndef __COMPONENT_H
#define __COMPONENT_H

  // SCCS ID follows: will compile to place date/time stamp in
  // object file

  static char COMPONENT_h_SccsId[] = "@(#)component.h   1.3\t9/16/91";

// Contents ----------------------------------------------------
  //
  //   COMPONENT
  //
  // Description
  //
  //   Defines class COMPONENT.
  //
  // End ---------------------------------------------------------

  // Implementation Dependencies --------------------------------
```

```
#include <Type.h>
#include <List.h>
#include <Reference.h>
#include "ReferenceMacros.h"
#include <stream.hxx>

 // End Implementation Dependencies ----------------------------

 // Interface Dependencies -------------------------------------

#ifndef __TEXT_OBJECT_H
#include "text_object.h"
#endif


 // End --------------------------------------------------------

 TypeCheckReference(TextObjDictReference, Reference, List);

// Class //

 class COMPONENT : public Object
{
 private :

  TextObjDictReference text_object_list;

 public:

 COMPONENT();
 COMPONENT(APL *theAPL);
 virtual void Destroy(Boolean abort = FALSE);
 virtual Type *getDirectType();
 void getComponentNames();
 Boolean getComponentSource(char*);
 void addTextObject(TEXT_OBJECT *);
 Boolean getPSfile(char*);
 Boolean getGRAPHfile(char*);
 Boolean getSPECfile(char*);
 Boolean getIMPfile(char*);
 Boolean getSOURCEfile(char*);
 char  *getTEXTPtr(char*);
 ~COMPONENT() { Destroy(FALSE); };
```

```
};

// Description ----------------------------------------------
//
//   Defines an COMPONENT class.  The class COMPONENT is a derived
//   class of Object.
//
// Constructor
//
//   COMPONENT
//
//   Constructs an COMPONENT object
//
//   COMPONENT --APL
//
//   ONTOS required constructor
//
// Public Members
//
//   Destroy
//
//   ONTOS heap mangagement method.
//
//   getDirectType
//
//   Return the ONTOS Type of class COMPONENT.
//
//   getComponentNames
//
//   Display the file names of the file contained in the COMPONENT node
//
//   getComponentSource
//
//   Output the contents of an COMPONENT node to files.
//
//   addTextObject
//
//   Inserts a text_object into the COMPONENT node.
//
//   getPSfile
//
//   Output the .ps file contained in the COMPONENT node.
//
//   getGRAPHfile
```

```
//
//   Output the .graph file contained in the COMPONENT node.
//
//   getSPECfile
//
//   Output the .spec file contained in the COMPONENT node.
//
//   getIMPfile
//
//   Output the .imp file contained in the COMPONENT node.
//
//   getSOURCEfile
//
//   Output the .a file contained in the COMPONENT node.
//
//   ~COMPONENT
//
//   Destructor for the COMPONENT class.
//
// End -------------------------------------------------------

#endif // __COMPONENT_H
```

```
// File Header ------------------------------------------------------------
 //.............:
 //.Filename.....: component.cxx
// Date        : 9/16/91
// Author      : Garry Lewis
//             : Drew Dwyer
// Modified by  : Michael D. O'Loughlin
// Date        : 6/18/92
// Modifications : Extensive modifications have been conducted on almost all of the
/*              original code written by Dwyer and Lewis.  Every function that
                accesses an attribute of an object or the entire object has had
                to be modified. The original code is not even recognizable in
                some functions.
                Because the functionality of the design database system changed
                completely from what Dwyer and Lewis developed each individual
                modification has not been documented. Because of the massive
                changes required I discarded a lot of obsolete code and build a
                new system on the base Dwyer and Lewis had established.
*/
// Compiler     : Glockenspiel C++ 2.1
//
// End header comments ----------------------------------------------------



// SCCS ID follows: will compile to place date/time stamp in
// object file

static char COMPONENT_cxx_SccsId[] = "@(#)component.cxx 1.3\t9/16/91";

// Contents ---------------------------------------------------
//
//   COMPONENT::COMPONENT          ONTOS constructor
//   COMPONENT::COMPONENT          constructor
//   COMPONENT::getDirectType
//   COMPONENT::Destroy
//   COMPONENT::getComponentNames
//   COMPONENT::getComponentSource
//   COMPONENT::addTextObject
//   COMPONENT::getPSfile
//   COMPONENT::getGRAPHfile
//   COMPONENT::getSPECfile
//   COMPONENT::getIMPfile
//   COMPONENT::getSOURCEfile
```

```
//
// Description
//
//   Implementation of class COMPONENT member functions.
//
// End ------------------------------------------------------------



// TURN DEBUG-TRACE ON OR OFF

#define DEBUG

#include "debug.h"


// Implementation Dependencies --------------------------------
#ifndef __DDBDEFINES_H
#include "ddbdefines.h"
#endif

#include <Object.h>
#include <GlobalEntities.h>

extern "C--"
{
#include <strings.h>
}

#ifndef __TRACER_H
#include "tracer.h"
#endif

#ifndef __COMPONENT_H
#include "component.h"
#endif

// End Implementation Dependencies------------------------------

extern Type *COMPONENT_OType;
extern Type *TEXT_OBJECT_OType;

// ONTOS required constructor //
```

```
COMPONENT::COMPONENT(APL *theAPL):(theAPL)
{
};

// Constructor //

COMPONENT::COMPONENT()

    // Summary ---------------------------------------------------
    //
    //   Constructs a persistent COMPONENT object.
    //
    // Parameter
    //
    //   None
    //
    // Functional description
    //
    //   Creates a list to hold text objects, then reset a reference
    //   to point to the list.
    //
    // End -------------------------------------------------------
{
 initDirectType(COMPONENT_OType);

 List *newTextObjList = new List(TEXT_OBJECT_OType);
 newTextObjList -> putObject();
 text_object_list.Reset(newTextObjList, this);
 putObject();
};
// End Constructor COMPONENT::COMPONENT//

// Member Function //

Type *COMPONENT::getDirectType()

    // Summary ---------------------------------------------------
    //
    //   returns the ONTOS Type for the COMPONENT class.
    //
    // Return value
    //
    //   A pointer to an ONTOS Type.
    //
```

```
    // End ---------------------------------------------------------

{
 return COMPONENT_OType;
};

// End Member Function COMPONENT::getDirectType //

// Member Function //

void COMPONENT::Destroy(Boolean aborted)
{
 if(aborted)
   {
    Object::Destroy(aborted);
   }
}

// End Member Function COMPONENT::Destroy ----------------------------

// Member Function //

void COMPONENT::getComponentNames()

    // Summary -----------------------------------------------------
    //
    //   Displays the name of each component in an COMPONENT object.
    //
    // Parameter
    //
    //   None
    //
    // Functional description
    //
    //   We get a pointer to the list and then create an iterator
    //   for the list.  We iterate over each text object and
    //   display the contents of the name field.
    //
    //
    // End ---------------------------------------------------------
{
 List *my_list = (List*)text_object_list.Binding(this);
 ListIterator my_iterator(my_list);
 TEXT_OBJECT *the_text_object;
```

```
 while(my_iterator.moreData())
  {
   the_text_object = (TEXT_OBJECT*)(Entity*)my_iterator();
   the_text_object -> displayFileName();
  }
}

// End Member Function COMPONENT::getComponentNames ----------

// Member Function //

Boolean COMPONENT::getComponentSource(char *fileMode)
{
 List *my_list = (List*)text_object_list.Binding(this);
 ListIterator my_iterator(my_list);
 TEXT_OBJECT *the_text_object;
 Boolean write_failed = FALSE;
 while(my_iterator.moreData())
  {
   the_text_object = (TEXT_OBJECT*)(Entity*)my_iterator();
   if (!the_text_object -> rebuildTextFile(fileMode))
    write_failed = TRUE;
  }
 if (write_failed)
  return FAILED;
 else
  return SUCCESS;
}

// End Member Function COMPONENT::getComponentSource ----------

// Member Function //

void COMPONENT::addTextObject(TEXT_OBJECT *my_text_object)
{
 List *my_list = (List*)text_object_list.Binding(this);
 my_list -> Insert(my_text_object);
 my_list -> putObject();
 putObject();
}

// End Member Function COMPONENT::addTextObject ----------
```

```
// Member Function //

Boolean COMPONENT::getPSfile(char *fileMode)
{
 List *my_list = (List*)text_object_list.Binding(this);
 ListIterator my_iterator(my_list);

 while(my_iterator.moreData())
  {
   TEXT_OBJECT *the_text_object = (TEXT_OBJECT*)(Entity*)my_iterator();
   char *the_file_name = the_text_object->getFileName();
   the_file_name=(the_file_name +
            (strlen(the_text_object->getFileName())-LENGTH_PS_EXT));
   if(strcmp(the_file_name,PS_EXT)==0)
    {
     if (the_text_object->rebuildTextFile(fileMode));
     return SUCCESS;
    }
  }
}

// End Member Function COMPONENT::getPSfile ----------------------

// Member Function //

Boolean COMPONENT::getSPECfile(char *fileMode)
{

 List *my_list = (List*)text_object_list.Binding(this);
 ListIterator my_iterator(my_list);
 while(my_iterator.moreData())
  {
   TEXT_OBJECT *the_text_object = (TEXT_OBJECT*)(Entity*)my_iterator();
   char *the_file_name = the_text_object->getFileName();
   the_file_name=(the_file_name +
            (strlen(the_text_object->getFileName())-LENGTH_SPEC_EXT));
   if(strcmp(the_file_name,SPEC_EXT)==0)
    {
     if (the_text_object->rebuildTextFile(fileMode))
      return SUCCESS;
     else
      return FAILED;
    }
  }
```

```
}

// End Member Function COMPONENT::getSPECfile ----------------------

// Member Function //

Boolean COMPONENT::getGRAPHfile(char *fileMode)
{
 List *my_list = (List*)text_object_list.Binding(this);
 ListIterator my_iterator(my_list);
 while(my_iterator.moreData())
  {
   TEXT_OBJECT *the_text_object = (TEXT_OBJECT*)(Entity*)my_iterator();
   char *the_file_name = the_text_object->getFileName();
   the_file_name=(the_file_name +
            (strlen(the_text_object->getFileName())-LENGTH_GRAPH_EXT));
   if(strcmp(the_file_name,GRAPH_EXT)==0)
    {
     if (the_text_object->rebuildTextFile(fileMode))
      return SUCCESS;
     else
      return FAILED;
    }
  }
}

// End Member Function COMPONENT::getGRAPHfile ----------------------

// Member Function //

Boolean COMPONENT::getIMPfile(char *fileMode)
{
 List *my_list = (List*)text_object_list.Binding(this);
 ListIterator my_iterator(my_list);
 while(my_iterator.moreData())
  {
   TEXT_OBJECT *the_text_object = (TEXT_OBJECT*)(Entity*)my_iterator();
   char *the_file_name = the_text_object->getFileName();
   the_file_name=(the_file_name +
            (strlen(the_text_object->getFileName())-LENGTH_IMP_EXT));
   if(strcmp(the_file_name,IMP_EXT)==0)
    {
     if (the_text_object->rebuildTextFile(fileMode))
      return SUCCESS;
```

```
      else
        return FAILED;
      }
    }
}

// End Member Function COMPONENT::getIMPfile ----------------------

// Member Function //

Boolean COMPONENT::getSOURCEfile(char *fileMode)
{
 List *my_list = (List*)text_object_list.Binding(this);
 ListIterator my_iterator(my_list);
 while(my_iterator.moreData())
   {
    TEXT_OBJECT *the_text_object = (TEXT_OBJECT*)(Entity*)my_iterator();
    char *the_file_name = the_text_object->getFileName();
    the_file_name=(the_file_name +
              (strlen(the_text_object->getFileName())-LENGTH_SOURCE_EXT));
    if(strcmp(the_file_name.SOURCE_EXT)==0)
      {
       if (the_text_object->rebuildTextFile(fileMode))
        return SUCCESS;
       else
        return FAILED;
      }
   }
}
// End Member Function COMPONENT::getSOURCEfile ----------------------

char *COMPONENT::getTEXTPtr(char *TextType)
{

 List *my_list = (List*)text_object_list.Binding(this);
 ListIterator my_iterator(my_list);
 while(my_iterator.moreData())
   {
    TEXT_OBJECT *the_text_object = (TEXT_OBJECT*)(Entity*)my_iterator();
    char *the_file_name = the_text_object->getFileName();
    char *the_file=(the_file_name +
              (strlen(the_text_object->getFileName())- (strlen(TextType))));
    if(strcmp(the_file,TextType)==0)
      {
```

```
        return the_text_object->text();
      }
    }
  return (char *)0;
}

// End Member Function COMPONENT::getSPECfile ----------
```

```
// File Header ---------------------------------------------------------
 //.............:
 //.Filename.....: conffunc.h
// Date        : 9/16/91
// Author      : Garry Lewis
//             : Drew Dwyer
// Modified by  : Michael D. O'Loughlin
// Date        : 6/18/92
// Modifications : Extensive modifications have been conducted on almost all of the
/*              original code written by Dwyer and Lewis.  Every function that
               accesses an attribute of an object or the entire object has had
               to be modified. The original code is not even recognizable in
               some functions.
               Because the functionality of the design database system changed
               completely from what Dwyer and Lewis developed each individual
               modification has not been documented. Because of the massive
               changes required I discarded a lot of obsolete code and build a
               new system on the base Dwyer and Lewis had established.
*/
// Compiler     : Glockenspiel C++ 2.1
//
// End header comments -------------------------------------------------



#ifndef __CONFFUNC_H
#define __CONFFUNC_H

#ifndef __CONFIGURATION_H
#include "configuration.h"
#endif

 // SCCS ID follows: will compile to place date/time stamp in
 // object file

 static char conffunc_h_SccsId[] = "@(#)conffunc.h     1.3\t9/16/91";

// Contents ------------------------------------------------------
 //
 //   Prototypes for functions related to manipulating
 //   configurations.
 //
 // End -----------------------------------------------------------
```

```
void list_conf_func(int, char *);
void dump_conf_summary_func(int,char *,char *);
void get_latest_conf_func(int, char *);
void attach_vobject_to_conf_func(int,char *, char *, char *, char *, char *);
void update_conf_name_func(int, char*, char*, char*);
void get_conf_desc_func(int, char*, char*);
void insert_conf_func(int, char*, char*, char*, char*);
void update_conf_desc_func(char*, char*, char*);
void get_conf_manager_func(int, char*, char*);
void update_conf_manager_func(int, char*, char*,char*);
void get_conf_date_func(int, char*, char*);
void post_conf_log_func(char*, char*, char*);
void get_conf_log_func(int, char*, char*);
//void dump_conf_operators_func(int,char *,char *);
void dump_conf_operators_func(int,char *,char *,char *);
void add_conf_operators_func(int,char *,char *);
void release_conf_lock_func(int,char *,char *);
void long_list_conf_operators_func(int,char *,char *);
void list_conf_operators_func(int,char *,char *);
void list_conf_default_operator_func(int,char *,char *);
CONFIGURATION* getConfiguration(char*, char*);
// Description ----------------------------------------------------
 //
 //   list_conf_func
 //
 //   Provides a list of configurations associated with a
 //   designated prototype.
 //
 //   dump_conf_summary_func
 //
 //   Provides summary information of a configuration to
 //   to include: creation date, manager, version number and
 //   default VOBJECT name, and a description.
 //
 //   get_latest_conf_func
 //
 //   Provides the name of the most recently added configuration.
 //
 //   attach_vobject_to_conf_func
 //
 //   Attaches an instance of the VOBJECT class to a configuration.
 //
 //   update_conf_name_func
 //
```

```
//  Changes the name of a designated configuration.
//
//  get_conf_desc_func
//
//  Provides a description of the designated configuration.
//
//  insert_conf_func
//
//  Creates a new configuration and binds it to a particular
//  prototype.
//
//  update_conf_desc_func
//
//  Updates the description text of a designated configuration.
//
//  get_conf_manager_func
//
//  Provides a configuration manager's name.
//
//  update_conf_manager_func
//
//  Changes a configuration manager's name.
//
//  get_conf_date_func
//
//  Provides the creation date of a given configuration.
//
//  post_conf_log_func
//
//  Adds a timestamped log entry (translated char string)
//  to a configuration.
//
//  get_conf_log_func
//
//  Provides the text of a configuration log.
//
//  dump_conf_operators_func
//
//  Write a copy of the operators in a designated configuration
//  to disk.
//
//  add_conf_operators_func
//
//  Add operator from disk to a particular configuration in the
```

```
//   database.
//
//   release_conf_lock_func
//
//   Provide a method to release locked operators in a given
//   configuration.
//
//   long_list_conf_operators_func
//
//   List all the children's names and version numbers of a
//   given configuration.
//
//   list_conf_operators_func
//
//   List the immediate children's name and version number of
//   a given configuration.
//
//   list_conf_default_operator_func
//
//   Provides the name and version number of the default VOBJECT
//   assigned to the configuration.
//
// End Description ----------------------------------------------

#endif // __CONFFUNC_H


// File Header ---------------------------------------------------------------
 //.............:
 //.Filename.....: conffunc.cxx
// Date         : 9/16/91
// Author       : Garry Lewis
//              : Drew Dwyer
// Modified by  : Michael D. O'Loughlin
// Date         : 6/18/92
// Modifications : Extensive modifications have been conducted on almost all of the
/*              original code written by Dwyer and Lewis.  Every function that
                accesses an attribute of an object or the entire object has had
                to be modified. The original code is not even recognizable in
                some functions.
                Because the functionality of the design database system changed
                completely from what Dwyer and Lewis developed each individual
                modification has not been documented. Because of the massive
                changes required I discarded a lot of obsolete code and build a
```

```
            new system on the base Dwyer and Lewis had established.
*/
// Compiler     : Glockenspiel C++ 2.1
//
// End header comments ------------------------------------------------



// SCCS ID follows: will compile to place date/time stamp in
// object file

static char conffunc_cxx_SccsId[] = "@(#)conffunc.cxx   1.3\t9/16/91";

// Contents -----------------------------------------------
//
//   list_conf_func
//   dump_conf_summary_func
//   get_latest_conf_func
//   attach_vobject_to_conf_func
//   update_conf_name_func
//   get_conf_desc_func
//   insert_conf_func
//   update_conf_desc_func
//   get_conf_manager_func
//   update_conf_manager_func
//   get_conf_date_func
//   post_conf_log_func
//   get_conf_log_func
//   dump_conf_operators_func
//   add_conf_operators_func
//   release_conf_lock_func
//   long_list_conf_operators_func
//   list_conf_operators_func
//   list_conf_default_operator_func
//
// End ----------------------------------------------------

// Implementation Dependencies ----------------------------------

// TURN ON TRACE-DEBUG
#define DEBUG
#include "debug.h"
```

```
#include <stream.hxx>
#include <Directory.h>
#include "My_String.h"

extern "C--"
{
#include <sys/time.h>
#include <sys/types.h>
#include <stdlib.h>
}

#ifndef __DIRECTORY_H
#include "directory.h"
#endif

#ifndef __TREE_H
#include "tree.h"
#endif

#ifndef __TREENODE_H
#include "treenode.h"
#endif

#ifndef __PROTOTYPE_H
#include "prototype.h"
#endif

#ifndef __THREAD_H
#include "thread.h"
#endif

#ifndef __CONFIGURATION_H
#include "configuration.h"
#endif

#ifndef __CONFFUNC_H
#include "conffunc.h"
#endif

#ifndef __VOBJECTFUNC_H
#include "vobjectfunc.h"
#endif

#ifndef __DDBDEFINES_H
```

```
#include "ddbdefines.h"
#endif

PROTOTYPE *protoPtr;
CONFIGURATION *configurationPtr;
extern THREAD *threadPtr;

// End Implementation Dependencies ----------------------------------

ifstream inputfile;

void list_conf_func(int number_arguments, char *arg1)
{
 TRACER("list_conf_func");
 char* prototype_name = (char*) (My_String(arg1) + My_String(PROTOTYPE_EXT));

 switch (number_arguments)
  {
  case 1:
   protoPtr = (PROTOTYPE*)OC_lookup(prototype_name);
   protoPtr -> listConfigurations();
   break;
  case 3:
  default:
   cerr << "<ERROR: extra arguments in list configurations>\n";
  }
}

void dump_conf_summary_func(int number_arguments, char *arg1, char *arg2)
{
 TRACER("dump_conf_summary_func");
 configurationPtr = getConfiguration(arg1, arg2);

 switch(number_arguments)
  {
  case 2:
   if(configurationPtr)
    {
     configurationPtr -> dumpConfigSummary();
    }
   break;
  default:
   cerr << "<ERROR: extra arguments in dump configuration summary>\n";
  }
```

```
}

void get_latest_conf_func(int number_arguments, char *arg1)
{
  TRACER("get_latest_conf_func");
 char* prototype_name = (char*) (My_String(arg1) + My_String(PROTOTYPE_EXT));

 switch (number_arguments)
   {
    case 1:
     protoPtr = (PROTOTYPE*)OC_lookup(prototype_name);
     protoPtr -> getDefaultConfigName();
     break;
    case 3:
    default:
     cerr << "<ERROR: extra arguments in get default configurations>\n";
   }
}

void attach_vobject_to_conf_func(int number_arguments, char *proto_name,
                        char *config_name, char *operator_name,
                        char *variationstr, char *versionstr)
{
 TRACER("attach_vobject_to_conf_func");

threadPtr = findThread(proto_name, operator_name);
if (!threadPtr)
 {
  return;
 }
configurationPtr = getConfiguration(proto_name, config_name);

if (configurationPtr)
 switch (number_arguments)
   {
   case 3:
      cerr << "CAO requires 5 arguments. Default values no longer valid.\n";
    break;
   case 5:
    V_OBJECT *variationPtr = threadPtr -> version(1);
    V_OBJECT *NewVariationPtr = variationPtr -> variation(atoi(variationstr));
    threadPtr = NewVariationPtr -> getThread();
       V_OBJECT *vobjectPtr;
       vobjectPtr = threadPtr->version(atoi(versionstr));
```

```
        configurationPtr->attachVobjecttoConfig(vobjectPtr);
        configurationPtr->putObject();
     break;
   default:
     cerr << "<ERROR: invalid number args for get vobject description>\n";
   }
}


void update_conf_name_func(int number_arguments, char *arg1, char *arg2, char *arg3)
{
 TRACER("update_conf_name_func");
 configurationPtr = getConfiguration(arg1, arg2);
 switch (number_arguments)
   {
   case 3:
    if (configurationPtr)
      {
       configurationPtr -> updateConfigName(arg3);
      }
    break;
   default:
    cerr << "<ERROR: invalid number args for update Config Name>\n";
   }
}


void get_conf_desc_func(int number_arguments, char *arg1, char *arg2)
{
 TRACER("get_conf_desc_func");
 configurationPtr = getConfiguration(arg1, arg2);

 switch (number_arguments)
   {
   case 2:
    if (configurationPtr)
      {
        configurationPtr -> getConfigDescription();
      }
    break;
   default:
    cerr << "<ERROR: invalid number args for get configuration description>\n";
   }
}
```

```
void insert_conf_func(int number_arguments, char *arg1, char *arg2, char *arg3, char *arg4)
{
 TRACER("insert_conf_func");
 configurationPtr = getConfiguration(arg1, arg2);

 switch (number_arguments)
  {
  case 2:
    if (!configurationPtr)
      {
       configurationPtr = new CONFIGURATION(arg2);
       protoPtr -> addConfiguration(configurationPtr);
      }
    else
      {
       cerr << "<ERROR: Configuration name " << arg2 << " already exists!>\n";
        return;
      }
   break;
  case 3:
   if (!configurationPtr)
    {
     configurationPtr = new CONFIGURATION(arg2, arg3);
     protoPtr -> addConfiguration(configurationPtr);
    }
   else
    {
     cerr << "<ERROR: Configuration name " << arg2 << " already exists!>\n";
     return;
    }
   break;
  case 4:
   if (!configurationPtr)
    {
     configurationPtr = new CONFIGURATION(arg2, arg3);
     inputfile.open(arg4, ios::in);
     if (!inputfile)
       {
        cerr << "<File with config description contents does not exist>\n"
             << "<Constructing configuration w/Name & Manager only>\n";
       }
     else
       {
```

```
        configurationPtr -> updateConfigDescription(arg4,inputfile);
       }
      protoPtr -> addConfiguration(configurationPtr);
      inputfile.close();
     }
    break;
  default:
    cerr << "<ERROR: invalid number args for insert configuration>\n";
  }
}


void update_conf_desc_func(char *arg1, char *arg2, char *arg3)
{
 TRACER("update_conf_desc_func");
 configurationPtr = getConfiguration(arg1, arg2);
 if (configurationPtr)
  {
   inputfile.open(arg3, ios::in);
   if (!inputfile)
    {
     cerr << "<File with config description contents not found>\n"
          << "<Aborting update configuration operation>\n";
    }
   else
    {
     configurationPtr -> updateConfigDescription(arg3, inputfile);
     inputfile.close();
    }
  }
}


void get_conf_manager_func(int number_arguments, char *arg1, char *arg2)
{
 TRACER("get_conf_manager_func");
 configurationPtr = getConfiguration(arg1, arg2);

 switch (number_arguments)
  {
  case 2:
    if (configurationPtr)
     {
      configurationPtr -> getConfigManager();
```

```
      }
     break;
   default:
    cerr << "<ERROR: invalid number args for insert configuration>\n";
   }
}


void update_conf_manager_func(int number_arguments, char *arg1, char *arg2, char *arg3)
{
 TRACER("update_conf_manager_func");
 configurationPtr = getConfiguration(arg1, arg2);

 switch (number_arguments)
  {
  case 3:
   if (configurationPtr)
    {
     configurationPtr -> updateConfigManager(arg3);
    }
   break;
  default:
   cerr << "<ERROR: invalid number args for update Config Manager>\n";
  }
}


void get_conf_date_func(int number_arguments, char *arg1, char *arg2)
{
 TRACER("get_conf_date_func");
 configurationPtr = getConfiguration(arg1, arg2);

 switch (number_arguments)
  {
  case 2:
   if (configurationPtr)
    {
     time_t systemtime = configurationPtr -> getConfCreationDate();
     cout << ctime(&systemtime) << "\n";
    }
   break;
  default:
   cerr << "<ERROR: invalid number args to get configuration creation date>\n";
  }
```

```
}


void post_conf_log_func(char *arg1, char *arg2, char *arg3)
{
 TRACER("post_conf_log_func");
 configurationPtr = getConfiguration(arg1, arg2);

 if (configurationPtr)
  {
   inputfile.open(arg3, ios::in);
   if (!inputfile)
    {
     configurationPtr -> addtoConfigLog(arg3);
    }
   else
    {
     configurationPtr -> addtoConfigLog(inputfile);
     inputfile.close();
    }
  }
}



void get_conf_log_func(int number_arguments, char *arg1, char *arg2)
{
 TRACER("get_conf_log_func");
 configurationPtr = getConfiguration(arg1, arg2);

 switch (number_arguments)
  {
  case 2:
   if (configurationPtr)
    {
     configurationPtr -> getConfigLog();
    }
  }
}


void dump_conf_operators_func(int number_arguments,char *proto_name,char *conf,
                              char *writeOption)
```

```
{
 TRACER("dump_conf_operators_func");
 configurationPtr = getConfiguration(proto_name, conf);

 switch (number_arguments)
  {
  case 3:
   if (configurationPtr)
    {
     V_OBJECT *vobjectPtr = configurationPtr->getDefaultVobject();
     if (vobjectPtr)
      {
        cout << vobjectPtr->getNodeName()
          << "  " << vobjectPtr->getVersionNumber()
          << vobjectPtr->getVariationNumber() <<"\n\n";
        Boolean file_operation_successful = FALSE;
        file_operation_successful =
           vobjectPtr -> checkoutCOMPONENTNode(writeOption); //(file_write_option);
        if (file_operation_successful)
          vobjectPtr -> dumpSubtree(writeOption);
        else
          cerr << "<Error checking out " << vobjectPtr ->getNodeName()
            << " Aborting dump_vobject_tree_func>\n";
      }
     else
      cerr << "<Error: No Vobject is attached to dump configuration>\n";
    }
   break;
  default:
   cerr << "<ERROR: invalid number args for dump configuration operators>\n";
  }
}

void add_conf_operators_func(int number_arguments,char *proto_name, char *conf)
{
 TRACER("add_conf_operators_func");
 configurationPtr = getConfiguration(proto_name, conf);

 switch (number_arguments)
  {
  case 2:
   if (configurationPtr)
    {
     V_OBJECT *vobjectPtr = configurationPtr->getDefaultVobject();
```

```
      if (vobjectPtr)
       {
        DIRECTORY *capsdirectory;
        capsdirectory = new DIRECTORY();
        char *operator_name = new char [strlen(vobjectPtr->getName())+1];
        strcpy(operator_name,vobjectPtr->getName());
        operator_name[strlen(operator_name) - 2] = '\0';
        capsdirectory->read_directory(operator_name);
        capsdirectory->updatetimestamp();
        TREENODE_linkedlist operatorList = capsdirectory->getOperatorList();
        TREENODE *rootnode = capsdirectory->find_treenode(operator_name);
        TREENODE *tree_root = new TREENODE(rootnode,NULL);
        TREE *workingtree = new TREE(tree_root, operator_name);
        workingtree->build_tree(tree_root,operatorList);
//      cerr << "CHECKIN--> " << operator_name << "\n";
        V_OBJECT *new_parent = (V_OBJECT *)0;
        new_parent= vobjectPtr->getParent();
        V_OBJECT *new_root = tree_root->checkin_node(new_parent);
        if (!new_root)
         {
          cerr << "<Error: Could not establish new_root in"
              << "add_conf_operators_func. Aborting.>\n";
          break;
         }
        new_root->setNodeName(tree_root->getname());
        tree_root->checkin_subtree(new_root);
       }
      else
        cerr << "<Error: No Vobject is attached to this configuration>\n";
     }
    break;
  default:
    cerr << "<ERROR: invalid number args for list configuration operators>\n";
  }
}

void release_conf_lock_func(int number_arguments, char *proto_name,
                  char *conf)
{
 TRACER("release_conf_lock_func");
 configurationPtr = getConfiguration(proto_name, conf);

 switch (number_arguments)
  {
```

```
  case 2:
   if (configurationPtr)
    {
     V_OBJECT *vobjectPtr = configurationPtr->getDefaultVobject();
     if (vobjectPtr)
      {
       if (vobjectPtr->releaseLock())
        {
         vobjectPtr->putObject();
         vobjectPtr -> releaseLockSubtree();
        }
      }
     else
      cerr << "<Error: No Vobject is attached to this configuration>\n";
    }
   break;
  default:
   cerr << "<ERROR: invalid number args for Release configuration lock>\n";
  }
}

void long_list_conf_operators_func(int number_arguments, char *proto_name,
                        char *conf)
{
 TRACER("long_list_conf_operators_func");
 configurationPtr = getConfiguration(proto_name, conf);

 switch (number_arguments)
  {
  case 2:
      if (configurationPtr)
       {
        V_OBJECT *vobjectPtr = configurationPtr->getDefaultVobject();
        if (vobjectPtr)
         {
          vobjectPtr -> longlistOperatorNames();
         }
        else
         cerr << "<Error: No Vobject is attached to this configuration>\n";
       }
   break;
  default:
   cerr << "<ERROR: invalid number args for list configuration operators>\n";
  }
```

```
}


void list_conf_operators_func(int number_arguments, char *proto_name,
                    char *conf)
{
 TRACER("list_conf_operators_func");
 configurationPtr = getConfiguration(proto_name, conf);

 switch (number_arguments)
  {
  case 2:
      if (configurationPtr)
       {
        V_OBJECT *vobjectPtr = configurationPtr->getDefaultVobject();
        if (vobjectPtr)
         {
          vobjectPtr -> listOperatorNames();
         }
        else
         cerr << "<Error: No Vobject is attached to this configuration>\n";
       }
    break;
  default:
   cerr << "<ERROR: invalid number args for list configuration operators>\n";
  }
}

void list_conf_default_operator_func(int number_arguments, char *proto_name,
                        char *conf)
{
 TRACER("list_conf_default_operator_func");
 configurationPtr = getConfiguration(proto_name, conf);

 switch (number_arguments)
  {
  case 2:
      if (configurationPtr)
       {
        V_OBJECT *vobjectPtr = configurationPtr->getDefaultVobject();
        if (vobjectPtr)
         {
          char *name=vobjectPtr->getName();
          name [strlen(name) - 2] = '\0';
```

```
            cout << name;
            int i=0;
            //************
            // Added following for statement for spacing...
            // ***********
            for (i=0;i<(PRINT_VERSION_LOCATION -
                    strlen(vobjectPtr->getName()));i++)
               cout << " ";
            cout <<  vobjectPtr->getVariationNumber();
            cout << " ";
            cout << vobjectPtr->getVersionNumber() << "\n";
           }
         else
           cerr << "<Error: No Vobject is attached to this configuration>\n";
        }
    break;
  default:
    cerr << "<ERROR: invalid number args for list "
     << "configuration default operator>\n";
  }
}


CONFIGURATION *getConfiguration(char *proto_name, char *config_name)
     {
      CONFIGURATION *configurationPtr = (CONFIGURATION *)0;
      char* prototype_name =
          (char*) (My_String(proto_name) + My_String(PROTOTYPE_EXT));
      protoPtr = (PROTOTYPE*)OC_lookup(prototype_name);

      if (protoPtr)
          {
           configurationPtr = (CONFIGURATION*)OC_lookup(config_name);
           if(!configurationPtr)
               {
                cerr << "<This Configuration name does not"
                     << " exist for this prototype.>\n";
               }
          }
      else
          {
           cerr << "<ERROR: Invalid Prototype name.>\n";
          }
      return configurationPtr;
```

}

```
// File Header ------------------------------------------------------------
 //.............:
 //.Filename.....: configuration.h
// Date        : 9/16/91
// Author      : Garry Lewis
//             : Drew Dwyer
// Modified by  : Michael D. O'Loughlin
// Date        : 6/18/92
// Modifications : Extensive modifications have been conducted on almost all of the
/*              original code written by Dwyer and Lewis.  Every function that
                accesses an attribute of an object or the entire object has had
                to be modified. The original code is not even recognizable in
                some functions.
                Because the functionality of the design database system changed
                completely from what Dwyer and Lewis developed each individual
                modification has not been documented. Because of the massive
                changes required I discarded a lot of obsolete code and build a
                new system on the base Dwyer and Lewis had established.
*/
// Compiler     : Glockenspiel C++ 2.1
//
// End header comments ----------------------------------------------------


#ifndef __CONFIGURATION_H
#define __CONFIGURATION_H

 // SCCS ID follows: will compile to place date/time stamp in object file

 static char configuration_h_SccsId[] = "@(#)configuration.h   1.3\t9/16/91";

// Contents ---------------------------------------------------
 //
 //   CONFIGURATION
 //
 // Description
 //
 //   Defines class CONFIGURATION
 //
 // End --------------------------------------------------------

 // Implementation Dependencies --------------------------------
```

```
#include <Object.h>
#include <Reference.h>
#include <Dictionary.h>
#include <stream.hxx>

 extern "C--"
{
#include <sys/time.h>
#include <sys/types.h>
}

#include "ReferenceMacros.h"

// End Implementation Dependencies ----------------------------

 // Interface Dependencies -------------------------------------

#ifndef __TEXT_OBJECT_H
#include "text_object.h"
#endif

#ifndef __VERSIONED_OBJECT_H
#include "versioned_object.h"
#endif

 // End Interface Dependencies ---------------------------------

 TypeCheckReference(V_ObjectReference, Reference, V_OBJECT);
TypeCheckReference(LogReference, Reference, TEXT_OBJECT);
TypeCheckReference(Desc2Reference, Reference, TEXT_OBJECT);

#define DEFAULT_MANAGER ""

// Class //

 class CONFIGURATION : public Object
{
 private:
 char config_status;
 char *config_manager;
 time_t ConfCreationDate;
 int config_num_vobjects;
 LogReference config_log_entry;
 Desc2Reference config_description;
```

```
 V_ObjectReference theVersioned_Object;

 public:
 CONFIGURATION(APL *);
 CONFIGURATION(char *name, char *manager=DEFAULT_MANAGER);
 virtual void Destroy(Boolean aborted=FALSE);
 virtual Type *getDirectType();
 void getConfigName();
 char *name();
 void getConfigStatus();
 void getConfigManager();
 void getConfigLog();
 void getConfigDescription();
 void dumpConfigSummary();
 void listConfigOperators();
 void updateConfigManager(char *new_config_manager);
 void updateConfigName(char *new_config_name);
 void updateConfigStatus(char new_config_status);
 void addtoConfigLog(char *new_log_entry);
 void addtoConfigLog(ifstream&);
 void updateConfigDescription(char *, ifstream& );
 V_OBJECT *CONFIGURATION::updateVobjectAttachment();
 void attachVobjecttoConfig(V_OBJECT*);
 time_t setConfCreationDate();
 time_t getConfCreationDate();
 V_OBJECT *getDefaultVobject();
 ~CONFIGURATION() { Destroy(FALSE); }
};



// Description ----------------------------------------------------
 //
 //   Defines a CONFIGURATION class.
 //
 // Constructor
 //
 //   configuration --APL
 //
 //   ONTOS required constructor
 //
 //   configuration
 //
 //   constructs a configuration object with the given name.
```

```
//  and manager.
//
// Public Members
//
//  destroy
//
//  Used to cleanup memory during deletion and transaction aborts.
//
//  getDirectType
//
//  Returns the ONTOS type for this class.
//
//  getConfigName;
//
//  Sends the configuration name to standard out.
//
//  name
//
//  Returns a pointer to the configuration name.
//
//  getConfigStatus
//
//  Sends the configuration status to standard out.
//
//  getConfigManager
//
//  Sends the manager's name for this particular configuration.
//
//  getConfigLog
//
//  Sends the configuration log to standard out.
//
//  getConfigDescription
//
//  Sends the configuration description to standard output.
//
//  dumpConfigSummary
//
//  Provides name, version number of root vobject , date and
//  description of configuration.
//
//  listConfigOperators
//
//  list the name of component operators in a configuration.
```

```
//
//   updateConfigManager
//
//   Changes the manager's name for this configuration.
//
//   updateConfigName
//
//   Changes the configuration name.
//
//   updateConfigStatus
//
//   Changes the configuration status field.
//
//   addtoConfigLog
//
//   A log to maintain a history of the configuration.
//
//   updateConfigDescription
//
//   Replaces the existing description if one exist or adds a new description.
//
//   attachVobjecttoConfig
//
//   Adds a versioned object to configuration.
//
//   setConfCreationDate
//
//   Time stamp this object with the current system time.
//
//   getConfCreationDate
//
//   Displays the time an instance of this class was created.
//
//   getDefaultVobject
//
//   Returns a pointer to the attach vobject.
//
//   ~configuration
//
//    A destructor for the configuration class.
//
// End ----------------------------------------------------------------------

#endif // __CONFIGURATION_H
```

```
// File Header ------------------------------------------------------------
 //.............:
 //.Filename.....: configuration.cxx
// Date        : 9/16/91
// Author      : Garry Lewis
//             : Drew Dwyer
// Modified by  : Michael D. O'Loughlin
// Date        : 6/18/92
// Modifications : Extensive modifications have been conducted on almost all of the
/*              original code written by Dwyer and Lewis.  Every function that
                accesses an attribute of an object or the entire object has had
                to be modified. The original code is not even recognizable in
                some functions.
                Because the functionality of the design database system changed
                completely from what Dwyer and Lewis developed each individual
                modification has not been documented. Because of the massive
                changes required I discarded a lot of obsolete code and build a
                new system on the base Dwyer and Lewis had established.
*/
// Compiler     : Glockenspiel C++ 2.1
//
// End header comments ----------------------------------------------------



// SCCS ID follows: will compile to place date/time stamp in
// object file

static char configuration_cxx_SccsId[] = "@(#)configuration.cxx 1.3\t9/16/91";

// Contents ---------------------------------------------------
//
//   CONFIGURATION::CONFIGURATION          ONTOS constructor
//   CONFIGURATION::CONFIGURATION          new instance
//   CONFIGURATION::Destroy
//   CONFIGURATION::getDirectType
//   CONFIGURATION::getConfigName
//   CONFIGURATION::name
//   CONFIGURATION::getConfigStatus
//   CONFIGURATION::getConfigManager
//   CONFIGURATION::getConfigLog
//   CONFIGURATION::getConfigDescription
//   CONFIGURATION::dumpConfigSummary
//   CONFIGURATION::listConfigOperators
```

```
//   CONFIGURATION::updateConfigManager
//   CONFIGURATION::updateConfigName
//   CONFIGURATION::updateConfigStatus
//   CONFIGURATION::addtoConfigurationLog -- string
//   CONFIGURATION::addtoConfigurationLog -- file
//   CONFIGURATION::updateConfigDescription
//   CONFIGURATION::attachVobjecttoConfig
//   CONFIGURATION::setConfCreationDate
//   CONFIGURATION::getConfCreationDate
//   CONFIGURATION::getDefaultVobject
//   CONFIGURATION::~CONFIGURATION
//
// Description
//
//   Implementation of class CONFIGURATION member functions.
//
// End ----------------------------------------------------

// Implementation Dependencies ---------------------------------

// TURN ON DEBUG-TRACE
#define DEBUG
#include "debug.h"


#include <GlobalEntities.h>
#include <Directory.h>
#include <stream.hxx>

extern "C--"
{
#include <string.h>
}

// End Implementation Dependencies ----------------------------

// Interface Dependencies -------------------------------------

#ifndef __CONFIGURATION_H
#include "configuration.h"
#endif

#ifndef __DDBDEFINES_H
#include "ddbdefines.h"
```

```
#endif

// End Interface Dependencies ---------------------------------

extern Type *V_OBJECT_OType;
extern Type *CONFIGURATION_OType;

// ONTOS required constructor //

CONFIGURATION::CONFIGURATION(APL *theAPL) : (theAPL)
{
};

// New Instance Constructors //

CONFIGURATION::CONFIGURATION(char *name,
                    char *manager):(name)

   // Summary ---------------------------------------------------------
   //
   //   Constructs a persistent CONFIGURATION object.  This object
   //   contains management (header information) about a CONFIGURATION
   //   and a select group of modules in the configuration.
   //
   // Parameter
   //
   //   name
   //
   //   A pointer to a character string.
   //
   //   manager
   //
   //   A pointer to a character string.
   //
   // Functional Description
   //
   //   Copies the manager's name into private data member. Initializes
   //   the description and log entry to null and creates a dictionary to
   //   hold the configuration modules.
   //
   // End -------------------------------------------------------------

{
 initDirectType(CONFIGURATION_OType);
```

```
 config_manager = new char[strlen(manager)+1];
 strcpy(config_manager, manager);
 config_status = 'A';
 config_num_vobjects = 0;
 ConfCreationDate = setConfCreationDate();
 config_description.initToNull();
 config_log_entry.initToNull();
 theVersioned_Object.initToNull();

 putObject();

}
// End Constructor for CONFIGURATION::CONFIGURATION

// Member Functions //

void CONFIGURATION::Destroy(Boolean aborted)
{
 delete config_manager;
 if (aborted)
   {
    Object::Destroy(aborted);
   }
}

Type *CONFIGURATION::getDirectType()
{
 return CONFIGURATION_OType;
}

void CONFIGURATION::getConfigName()
{
 Directory *directory;
 char *name;

 if(!this)
   {
    cerr << "<ERROR: cannot get the name of a null CONFIGURATION>\n";
    return;
   }
 name = Name();
 OC_getNameComponents(name, &directory, &name);
 cout << name << "\n";
}
```

```
char *CONFIGURATION::name()
{
 Directory *directory;
 char *name;

 name = Name();
 OC_getNameComponents(name, &directory, &name);
 return name;
}

void CONFIGURATION::getConfigStatus()
{
 if(!this)
   {
    cerr << "<ERROR: cannot get the Status of a null Configuration>\n";
    return;
   }
 cout << config_status << "\n";
}

void CONFIGURATION::getConfigManager()
{
 if(!this)
   {
    cerr << "<ERROR: cannot get the Manager of a null Configuration>\n";
    return;
   }
 cout << config_manager << "\n";
}

void CONFIGURATION::getConfigLog()

{
 if(!this)
   {
    cerr << "<ERROR: cannot dump the Log of a null Configuration>\n";
    return;
   }
 if (!config_log_entry)
   {
    cerr << "<Cannot display an empty log>\n";
    return;
   }
```

```
 else
  {
   TEXT_OBJECT* myTextObject = (TEXT_OBJECT*)config_log_entry.Binding(this);
   myTextObject -> text(cout);
  }
}

void CONFIGURATION::getConfigDescription()
{
 if(!this)
  {
   cerr << "<ERROR: cannot get the description of a null Configuration>\n";
   return;
  }
 if (!config_description)
  {
   cerr << "<This configuration does not contain a description>\n";
   return;
  }
 else
  {
   TEXT_OBJECT* myTextObject = (TEXT_OBJECT*)config_description.Binding(this);
   myTextObject -> text(cout);
  }
}


void CONFIGURATION::dumpConfigSummary()
{
 TRACER("CONFIGURATION::dumpConfigSummary");
 int i=0;
 cout << ctime(&ConfCreationDate);
 cout << "  ";
 getConfigManager();
 if(!theVersioned_Object)
  {
   cout << " ";
   for (i=0;i<PRINT_VERSION_LOCATION -strlen("VOBJECT Name: NONE ASSIGNED ");i++)
    cout << ' ';
   cout << " ";
   cout << "\n";
  }
 else
  {
```

```
    V_OBJECT *vobjectPtr = (V_OBJECT*) theVersioned_Object.Binding(this);
    char *name = vobjectPtr ->getName();
    name[strlen(name)-2] = '\0';
    cout << name;
    for (i=0;i < PRINT_VERSION_LOCATION - strlen(vobjectPtr->getName()) -
            strlen("VOBJECT Name: ")   ;  i++) cout << ' ';
    cout << " ";
    cout << vobjectPtr -> getVariationNumber() << " ";
    cout << " ";
    cout << vobjectPtr -> getVersionNumber() << "\n";
   }
 getConfigDescription();
}

void CONFIGURATION::listConfigOperators()
{
 if(!theVersioned_Object)
   {
    cerr << "This configuration does not contain a v_object";
   }
 else
   {
    V_OBJECT *theVObjectPtr =
     (V_OBJECT*) theVersioned_Object.Binding(this);
    theVObjectPtr -> getVObjName();
    theVObjectPtr -> listOperatorNames();
   }
}

void CONFIGURATION::updateConfigManager(char *new_config_manager)
{
 if(!this)
   {
    cerr << "<ERROR: cannot change the manager of a null CONFIGURATION>\n";
    return;
   }
 if(config_manager)
   {
    strcpy(config_manager, "");
   }
 config_manager = new char[strlen(new_config_manager)+1];
 strcpy(config_manager, new_config_manager);
 putObject();
}
```

```
void CONFIGURATION::updateConfigName(char *new_config_name)
{
 if(!this)
   {
    cerr << "<ERROR: cannot change the name of a NULL CONFIGURATION>\n";
    return;
   }

 Name(new_config_name);
}



void CONFIGURATION::updateConfigStatus(char new_config_status)
{
 if(!this)
   {
    cerr << "<ERROR: cannot change the status of a null CONFIGURATION>\n";
    return;
   }
 config_status = new_config_status;
}


void CONFIGURATION::addtoConfigLog(char *new_log_entry)
{
 if(!config_log_entry)
   {
    TEXT_OBJECT *textObjectPtr = new TEXT_OBJECT();
    textObjectPtr -> append(new_log_entry);
    config_log_entry.Reset(textObjectPtr, this);
   }
 else
   {
    TEXT_OBJECT *textObjectPtr =
     (TEXT_OBJECT*) config_log_entry.Binding(this);
    textObjectPtr -> append(new_log_entry);
   }
 putObject();
}

void CONFIGURATION::addtoConfigLog(ifstream& input_file_stream)
{
```

```
 if(!config_log_entry)
   {
    TEXT_OBJECT *textObjectPtr = new TEXT_OBJECT();
    textObjectPtr -> append(input_file_stream);
    config_log_entry.Reset(textObjectPtr, this);
   }
 else
   {
    TEXT_OBJECT *textObjectPtr =
     (TEXT_OBJECT*) config_log_entry.Binding(this);
    textObjectPtr -> append(input_file_stream);
   }
 putObject();
}



void CONFIGURATION::updateConfigDescription(char *fileName, ifstream& input_file_stream)
{
 if(!config_description)
   {
    TEXT_OBJECT *textObjectPtr = new TEXT_OBJECT();
    textObjectPtr -> append(fileName, input_file_stream);
    config_description.Reset(textObjectPtr, this);
   }
 else
   {
    TEXT_OBJECT *textObjectPtr =
     (TEXT_OBJECT*) config_description.Binding(this);
    textObjectPtr -> resetTheText();
    textObjectPtr -> append(fileName, input_file_stream);
   }
 putObject();
}

V_OBJECT *CONFIGURATION::updateVobjectAttachment()
{
 if (!this)
   {
    cerr << "<ERROR: cannot set the v_object of a null configuration\n";
    return NULL;
   }
 V_OBJECT *vobjectPtr = getDefaultVobject();
 if (vobjectPtr)
```

```
  {
   THREAD *threadPtr = (THREAD *)OC_lookup(vobjectPtr->getName());
   if (threadPtr)
    {
     vobjectPtr = threadPtr->current();
     theVersioned_Object.Reset(vobjectPtr,this);
     putObject();
    }
  }
 return vobjectPtr;
}

void CONFIGURATION::attachVobjecttoConfig(V_OBJECT *theV_Object)
{
 if (!this)
  {
   cerr << "<ERROR: cannot set the v_object of a null configuration\n";
   return;
  }
 if (!theV_Object)
  {
   cerr << "<ERROR: cannot give to a configuration a null v_object>\n";
  }
 theVersioned_Object.initToNull();
 theVersioned_Object.Reset(theV_Object, this);
}

//Member Function //

time_t CONFIGURATION::setConfCreationDate()
{
 time_t mytloc=0;
 time_t theTime;
 return theTime = time(mytloc);
}

// End ----------------------------------------------------


// Member Function //

time_t CONFIGURATION::getConfCreationDate()
{
 return ConfCreationDate;
```

```
}

// Member Function //

V_OBJECT * CONFIGURATION::getDefaultVobject()
{
 return (V_OBJECT *)(Entity *)theVersioned_Object.Binding(this);
}

// End -----------------------------------------------------

// end functions
```

```
 // File Header ------------------------------------------------------------
 //.............:
 //.Filename.....: ddbdefines.h
// Date         : 9/16/91
// Author       : Garry Lewis
//              : Drew Dwyer
// Modified by  : Michael D. O'Loughlin
// Date         : 6/18/92
// Modifications : Extensive modifications have been conducted on almost all of the
/*              original code written by Dwyer and Lewis.  Every function that
                accesses an attribute of an object or the entire object has had
                to be modified. The original code is not even recognizable in
                some functions.
                Because the functionality of the design database system changed
                completely from what Dwyer and Lewis developed each individual
                modification has not been documented. Because of the massive
                changes required I discarded a lot of obsolete code and build a
                new system on the base Dwyer and Lewis had established.
*/
// Compiler     : Glockenspiel C++ 2.1
//
// End header comments ----------------------------------------------------


#ifndef __DDBDEFINES_H
#define __DDBDEFINES_H

  // SCCS ID follows: will compile to place date/time stamp in object file

  static char ddbdefines_h_SccsId[] = "@(#)ddbdefines.h 1.3\t9/16/91";

// Contents ---------------------------------------------------
  //
  // Number Defines for Evaluations functions
  //
  // Description
  //
  // These #defines are all designed just to pass information back and forth
  // between the main program and the modules which evaluate the command line
  // for what function to run.  There are basically three types of function
  // arguments CONFIGURATION arguments -- beginning with a 'C', PROTOTYPE
  // arguments -- beginning with a 'P', and VOBJECT functions -- beginning
  // with a 'V'.  All arguments are exactly 3 characters in length and must
  // conform to one of the valid arguments in the designed interface.  All
```

```
// other arguments will be invalid and return <Invalid Function> to the
// Standard I/O.
//
//
// End -----------------------------------------------------

// Interface Dependencies ----------------------------------
//
// NONE
//
// End Interface Dependencies ------------------------------

struct DDBControlData
    {
    char *prototype_name;
    char *v_object_name;
    int variation;
    int version;
    };

#define PRINT_CONFIG_LOCATION 20
#define PRINT_VERSION_LOCATION 50
#define MAX_LINE_LENGTH 1024
#define COMMAND_TABLE_SIZE 50
#define SUCCESS TRUE
#define FAILED FALSE

#define LIST_PROTOTYPES  1
#define LONG_LIST_PROTOTYPES 81230
#define GET_PROTOTYPE_LEADER  2
#define GET_PROTOTYPE_DESCRIPTION  3
#define RETRIEVE_PROTOTYPE_DATE  5
#define INSERT_PROTOTYPE  6
#define UPDATE_PROTOTYPE_LEADER  7
#define UPDATE_PROTOTYPE_DESC  8
#define UPDATE_PROTOTYPE_NAME  9
#define GET_LATEST_CONFIGURATION  10
#define PROTOTYPE_VARIATION_LIST  11
#define PROTOTYPE_VAR_AND_VER_LIST  12
#define PROTOTYPE_VERSION_LIST  13
#define DUMP_PROTOTYPE_SUMMARY 987

#define LIST_CONFIGURATIONS  21
#define DUMP_CONFIGURATION_OPERATORS 91372
```

```
#define ADD_CONFIGURATION_OPERATORS 91378
#define LONG_LIST_CONFIGURATION_OPERATORS 92351
#define LIST_CONFIGURATION_DEFAULT_OPERATOR 6189
#define LIST_CONFIGURATION_OPERATORS 91375
#define UPDATE_CONFIGURATION_NAME  22
#define GET_CONFIGURATION_DESCRIPTION  23
#define INSERT_CONFIGURATION  24
#define UPDATE_CONFIGURATION_DESCRIPTION  25
#define GET_CONFIGURATION_MANAGER  26
#define UPDATE_CONFIGURATION_MANAGER  27
#define GET_CONFIGURATION_DATE  28
#define GET_CONFIGURATION_CHANGED  29
#define POST_CONFIGURATION_LOG  30
#define GET_CONFIGURATION_LOG  31
#define ATTACH_OPERATOR  32
#define DUMP_CONFIGURATION_SUMMARY 33
#define RELEASE_CONFIGURATION_LOCK 8124

#define LIST_OPERATORS  41
#define GET_VOBJECT_DESCRIPTION  42
#define GET_VOBJECT_DATE  43
#define GET_VOBJECT_VERSIONS  44
#define GET_VOBJECT_LOCK  45
#define GET_VOBJECT_VERSION  46
#define DUMP_VOBJECT_SUMMARY  47
#define GET_VOBJECT_POSTSCRIPT  48
#define GET_VOBJECT_GRAPH  49
#define GET_VOBJECT_IMPLEMENTATION  50
#define GET_VOBJECT_SPECIFICATION  51
#define GET_VOBJECT_SOURCE  52
#define UPDATE_VOBJECT_DESCRIPTION  53
#define ADD_NEW_VARIATION  57
#define ADD_VOBJECT_AND_SUBTREE  58
#define DUMP_VOBJECT_FILES  59
#define DUMP_VOBJECT_TREE  60
#define LONG_LIST_CHILDREN  61
#define LONG_LIST_PARENTS  62
#define LONG_LIST_OPERATORS  32981
#define RELEASE_SUBTREE_LOCK  8281
#define RELEASE_OPERATOR_LOCK  8992

#define HISTORICAL_TRAIL 80
#define ERROR_IN_EVALUATION 9999
```

```
#define LENGTH_PS_EXT 3
#define LENGTH_GRAPH_EXT 6
#define LENGTH_IMP_EXT 9
#define LENGTH_SPEC_EXT 10
#define LENGTH_SOURCE_EXT 2

#define PS_EXT ".ps"
#define GRAPH_EXT ".graph"
#define IMP_EXT ".imp.psdl"
#define SPEC_EXT ".spec.psdl"
#define SOURCE_EXT ".a"

#define DESIGN_DATABASE_DIRECTORY "^DesignDatabase"
#define PROTOTYPE_LIST          "PrototypeList"
#define LONG_LIST_PROTOTYPE_UPC  "PLL"
#define LONG_LIST_PROTOTYPE_LC   "pll"
#define LIST_PROTOTYPE_UPC       "PLN"
#define LIST_PROTOTYPE_LC        "pln"
#define INSERT_PROTOTYPE_UPC     "PIP"
#define INSERT_PROTOTYPE_LC      "pip"
#define PROTOTYPE_LIST_VARIATION_UPC  "PLV"
#define PROTOTYPE_LIST_VARIATION_LC   "plv"
#define PROTOTYPE_LIST_VERSION_UPC  "PVV"
#define PROTOTYPE_LIST_VERSION_LC   "pvv"
#define PROTOTYPE_LIST_ALL_UPC  "PVA"
#define PROTOTYPE_LIST_ALL_LC  "pva"

#define PROTOTYPE_EXT           ".prj"

#endif // __DDBDEFINES_H


// File Header ------------------------------------------------------------
 //.....:
 //Filename.....: directory.h
// Date        : 9/16/91
// Author      : Garry Lewis
//             : Drew Dwyer
// Modified by  : Michael D. O'Loughlin
// Date        : 6/18/92
// Modifications : Extensive modifications have been conducted on almost all of the
/*              original code written by Dwyer and Lewis.  Every function that
                accesses an attribute of an object or the entire object has had
                to be modified. The original code is not even recognizable in
```

```
            some functions.
            Because the functionality of the design database system changed
            completely from what Dwyer and Lewis developed each individual
            modification has not been documented. Because of the massive
            changes required I discarded a lot of obsolete code and build a
            new system on the base Dwyer and Lewis had established.
*/
// Compiler     : Glockenspiel C++ 2.1
//
// End header comments ---------------------------------------------------


#ifndef __DIRECTORY_H
#define __DIRECTORY_H

 // SCCS ID follows: will compile to place date/time stamp in
 // object file

 static char directory_h_SccsId[] = "@(#)directory.h   1.3\t9/16/91";

// Contents --------------------------------------------------
 //
 //   DIRECTORY HEADER
 //
 // Description
 //
 //   Defines class DIRECTORY.
 //
 // End -------------------------------------------------------

 // Interface Dependencies ------------------------------------

#ifndef __TREENODE_H
#include "treenode.h"
#endif

 // End Interface Dependencies --------------------------------

 class DIRECTORY
{
 private:
 TREENODE_linkedlist operator_nodes;
```

```
 public:
 DIRECTORY() {};
 void read_directory(char *root_oper);
 void updatetimestamp();
 TREENODE *find_treenode(char *);
 TREENODE_linkedlist getOperatorList();
};

// Description ----------------------------------------------
 //
 //   Defines class DIRECTORY.  Class DIRECTORY is a non-
 //   persistent class.
 //
 // Constructor
 //
 //   DIRECTORY
 //
 // Public Members
 //
 //   read_directory
 //
 //   Read a list of file from a directory defined by the environment
 //   variable PROTOTYPE, creates a corresponding list of operator nodes.
 //
 //   updatetimestamp
 //
 //   Updates the nodes time to reflect the time of the file
 //   most recently updated.
 //
 //   find_treenode
 //
 //   Find a given node in the list of operator nodes.
 //
 //   getOperatorList
 //
 //   Returns the operator node list.
 //
 // End Description ------------------------------------------

#endif   // header file
```

```
// File Header ------------------------------------------------------------
 //.............:
 //.Filename.....: directory.cxx
 //.SCCS ID......: 1.3
 //.Release No....: 1
 //.Date.........: 9/16/91
 //.Author.......: Garry Lewis
 //.............: Drew Dwyer
//.Modified by...: Michael D. O'Loughlin
//.Modifications.: The modifications made to this module correspond to the
//              modicications made to ....., as of .....
//              The modifications made to this module will, .....
//              The following operations were added or modified:
//                1.
//                2.
//                3.
//                4.
//              The above modifications were made on .....

 //.Compiler.....: Glockenspiel C++ 2.1
 //.............:
 // End header comments ---------------------------------------------------

// SCCS ID follows: will compile to place date/time stamp in
// object file

static char directory_cxx_SccsId[] = "@(#)directory.cxx 1.3\t9/16/91";

// Contents --------------------------------------------------
//
//   DIRECTORY::read_directory
//   DIRECTORY::updatetimestamp
//   DIRECTORY::find_treenode
//   DIRECTORY::getOperatorList
//
// Description
//
//   IMPLEMENTS class DIRECTORY CONSTRUCTORS.
//
// End -------------------------------------------------------------

// Interface Dependencies ------------------------------------

// TURN ON DEBUG_TRACE
```

```
#define DEBUG
#include "debug.h"
#include "My_String.h"
#include <stream.hxx>

extern "C--"
{
#include <stddef.h>
#include <stdlib.h>
#include <string.h>
#include <dirent.h>
#include <sys/stat.h>
#include <time.h>
}

#ifndef __DIRECTORY_H
#include "directory.h"
#endif

#ifndef __DDBDEFINES_H
#include "ddbdefines.h"
#endif
// ----------  End Interface Dependencies ------------------

extern char *dirNamePtr;

void  DIRECTORY::read_directory(char *root_oper)
{

 // ------------------------------------------------------------------
 // In body comment:
 //
 // Here I will create the list to hold those filenames that
 // contain (as a substring) the operator name.  This method
 // will also scan those files that do match the pattern input
 // and throw out those with the .ps, .graph, .imp.psdl, .spec.psdl
 // and .a files.  What I hope will remain is a list of only those
 // filenames which represent that operator node and it's
 // associated SUBTREE operator nodes.  We'll then turn around and
 // process the resulting list into an operator tree structure and
 // compare against the database schema for storage of the
 // .ps, graph, .spec.psdl, .imp.psdl, and .a text objects into
 //  the database as collected sets of COMPONENT objects.
```

```
//
// ------------------------------------------------------------
TRACER("DIRECTORY::read_directory");
DIR  *dirp;
struct dirent *capsdirent;
char *path[MAX_LINE_LENGTH];
char *pschk = NULL;
char *graphchk =NULL;
char *specchk = NULL;
char *impchk = NULL;
char *sourcechk =NULL;
TREENODE *operatornode =NULL;
char *filename = NULL;
strcpy(path,dirNamePtr);
dirp    = opendir(dirNamePtr);
int count=0;

TREENODE *temp;
TRACE(root_oper);
for (capsdirent = readdir(dirp); capsdirent != NULL;
    capsdirent = readdir(dirp))
 {
   filename=capsdirent->d_name;
   TRACE(filename);
   pschk = capsdirent ->d_name + strlen(capsdirent ->d_name) - 3;
   graphchk = capsdirent ->d_name + strlen(capsdirent ->d_name) - 6;
   specchk = capsdirent ->d_name + strlen(capsdirent ->d_name) - 10;
   impchk = capsdirent ->d_name + strlen(capsdirent ->d_name) - 9;
   sourcechk = capsdirent ->d_name + strlen(capsdirent ->d_name) - 2;

   if (strcmp(pschk,".ps")==0)
     {
      pschk[0] = '\0';
     }
   else if (strcmp(graphchk,".graph")==0)
     {
      graphchk[0] = '\0';
     }
   else if (strcmp(specchk,".spec.psdl")==0)
     {
      specchk[0] = '\0';
     }
   else if (strcmp(impchk,".imp.psdl")==0)
     {
```

```
        impchk[0] = '\0';
      }
    else if (strcmp(sourcechk,".a")==0)
      {
       sourcechk[0] = '\0';
      }


    if (strncmp(capsdirent ->d_name,root_oper,strlen(root_oper))==0)
      {
       TRACE(capsdirent->d_name);
       TRACE(root_oper);
       if (!(temp=find_treenode(capsdirent->d_name)))
         {
          operatornode = new TREENODE(capsdirent->d_name,NULL);
           TRACE("A NEW OPERATOR NODE IS BEING INSERTED");
           TRACE(capsdirent->d_name);
          operator_nodes.insert(operatornode);
         }
      }
   }
 closedir(dirp);
}

void DIRECTORY::updatetimestamp()
{
 DIR  *dirp;
 struct dirent *filep;

 struct stat timestats;

 char *psfilename = NULL;
 char *graphfilename = NULL;
 char *specfilename = NULL;
 char *impfilename = NULL;
 char *sourcefilename = NULL;
 char* path;
 char *node_name = NULL;
 TREENODE *node;
 long temptime = 0;
 long filetime = 0;
 dirp = opendir(dirNamePtr);
 slist_iterator OperatorPtr(operator_nodes);
```

```
while (node = OperatorPtr())
 {

  node_name = node->getname();

  psfilename    = (char*)(My_String(node_name) + My_String(".ps"));
  graphfilename  = (char*)(My_String(node_name) + My_String(".graph"));
  impfilename    = (char*)(My_String(node_name) + My_String(".imp.psdl"));
  specfilename   = (char*)(My_String(node_name) + My_String(".spec.psdl"));
  sourcefilename = (char*)(My_String(node_name) + My_String(".a"));

  filep = readdir(dirp);
  while ((filep != NULL) && (!(strcmp(filep->d_name,psfilename)==0)))
   {
    filep = readdir(dirp);
   }
  if ((filep != NULL) && (strcmp(filep->d_name,psfilename)==0))
   {
    path = (char *)(My_String(dirNamePtr) + My_String("/") +
        My_String(filep->d_name) );
    stat(path,&timestats);
    filetime = timestats.st_ctime;
    temptime = node->get_long_time();
    node->updatetimestamp(temptime < filetime ? filetime : temptime);
   }

  rewinddir(dirp);
  filep = readdir(dirp);
  while ((filep != NULL) && (!(strcmp(filep->d_name,graphfilename)==0)))
   {
    filep = readdir(dirp);
   }
  if ((filep != NULL) && (strcmp(filep->d_name,graphfilename)==0))
   {
    path = (char *)(My_String(dirNamePtr) + My_String("/") +
        My_String(filep->d_name) );
    stat(path,&timestats);
    filetime = timestats.st_ctime;
    temptime = node->get_long_time();
    node->updatetimestamp(temptime < filetime ? filetime : temptime);
   }

  rewinddir(dirp);
  filep = readdir(dirp);
```

```
while ((filep != NULL) && (!(strcmp(filep->d_name,impfilename)==0)))
 {
  filep = readdir(dirp);
 }
if ((filep != NULL) && (strcmp(filep->d_name,impfilename)==0))
 {
  path = (char *)(My_String(dirNamePtr) + My_String("/") +
      My_String(filep->d_name) );
  stat(path,&timestats);
  filetime = timestats.st_ctime;
  temptime = node->get_long_time();
  node->updatetimestamp(temptime < filetime ? filetime : temptime);
 }

rewinddir(dirp);
filep = readdir(dirp);
while ((filep != NULL) && (!(strcmp(filep->d_name,specfilename)==0)))
 {
  filep = readdir(dirp);
 }
if ((filep != NULL) && (strcmp(filep->d_name,specfilename)==0))
 {
  path = (char *)(My_String(dirNamePtr) + My_String("/") +
      My_String(filep->d_name) );
  stat(path,&timestats);
  filetime = timestats.st_ctime;
  temptime = node->get_long_time();
  node->updatetimestamp(temptime < filetime ? filetime : temptime);
 }

rewinddir(dirp);
filep = readdir(dirp);
while ((filep != NULL) && (!(strcmp(filep->d_name,sourcefilename)==0)))
 {
  filep = readdir(dirp);
 }
if ((filep != NULL) && (strcmp(filep->d_name,sourcefilename)==0))
 {
  path = (char *)(My_String(dirNamePtr) + My_String("/") +
      My_String(filep->d_name) );
  stat(path,&timestats);
  filetime = timestats.st_ctime;
  temptime = node->get_long_time();
  node->updatetimestamp(temptime < filetime ? filetime : temptime);
```

```
      }
    rewinddir(dirp);
   }
 closedir(dirp);
}


TREENODE *DIRECTORY::find_treenode(char *node_name)
{
 slist_iterator list_iterator(operator_nodes);
 TREENODE *tnode;
 while (tnode=list_iterator())
  if (strcmp(tnode->getname(),node_name)==0)
    return tnode;
 return NULL;
}


TREENODE_linkedlist DIRECTORY::getOperatorList()
{
 return operator_nodes;
}
```

```
// File Header ------------------------------------------------------------
 //.............:
 //.Filename.....: evaluation.cxx
// Date        : 9/16/91
// Author      : Garry Lewis
//             : Drew Dwyer
// Modified by  : Michael D. O'Loughlin
// Date        : 6/18/92
// Modifications : Extensive modifications have been conducted on almost all of the
/*              original code written by Dwyer and Lewis.  Every function that
                accesses an attribute of an object or the entire object has had
                to be modified. The original code is not even recognizable in
                some functions.
                Because the functionality of the design database system changed
                completely from what Dwyer and Lewis developed each individual
                modification has not been documented. Because of the massive
                changes required I discarded a lot of obsolete code and build a
                new system on the base Dwyer and Lewis had established.
*/
// Compiler     : Glockenspiel C++ 2.1
//
// End header comments ----------------------------------------------------



// SCCS ID follows: will compile to place date/time stamp in
// object file

static char evaluation_cxx_SccsId[] = "@(#)evaluation.cxx       1.3\t9/16/91";

// Contents ---------------------------------------------------
//
//   charupper
//   upper
//   evaluate_configuration_function
//   evaluate_historical_function
//   evaluate_vobject_function
//   evaluate_prototype_function
//
// Description
//
//   Defines FUNCTIONS FOR SWITCH STMT IN MAIN.
//
//   This information is required to evaluate the command
```

```
//   line input and reconstruct the proper commands internal
//   to the design database program.
//
//
// End ----------------------------------------------------

// Interface Dependencies ---------------------------------

#define DEBUG
#include "debug.h"

#ifndef __DDBDEFINES_H
#include "ddbdefines.h"
#endif

#include <stream.hxx>
extern "C--"
{
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
}

#ifndef __EVALUATION_H
#include "evaluation.h"
#endif

// End Interface Dependencies -----------------------------

char charupper(char c)
{
 return islower(c) ? (c-'a'+'A') : c;     // change char to upper case
};


char *upper(char *argument)
{
 int i;
 for (i=0; i<strlen(argument); i++)        // convert argument to upper case
   argument[i] = charupper(argument[i]);  // call charupper to convert each one
 return argument;
};

int evaluate_historical_function(char *function, int arguments)
```

```
{
 if (strcmp(function,"HTT")==0)
   {
    if (arguments<4 || arguments >4)
      {
       cerr << "<ERROR: Invalid number arguments for Historical Trail>\n";
       return ERROR_IN_EVALUATION;
      }
    else
     return HISTORICAL_TRAIL;
   }
}

int evaluate_configuration_function(char *function, int arguments)
{
 if (strcmp(function,"CLN")==0)
   {
    if (arguments<1 || arguments >1)
      {
       cerr << "<ERROR: Invalid number arguments for List Configurations>\n";
       return ERROR_IN_EVALUATION;
      }
    else
     return LIST_CONFIGURATIONS;              // tell main() to run List_Prototypes
   }
 else if (strcmp(function,"CUN")==0)
   {
    if (arguments<3 || arguments>3)
      {
       cerr << "<ERROR: Invalid number of arguments for Update Configuration Name>\n";
       return ERROR_IN_EVALUATION;
      }
    return UPDATE_CONFIGURATION_NAME;
   }
 else if (strcmp(function,"CGD")==0)
   {
    if (arguments<2 || arguments >2)
      {
       cerr << "<ERROR: Invalid number arguments for Get Configuration Description>\n";
       return ERROR_IN_EVALUATION;
      }
    return GET_CONFIGURATION_DESCRIPTION;
   }
 else if (strcmp(function,"CIC")==0)
```

```
  {
   if (!arguments>0)
    {
     cerr << "<ERROR: Not enough arguments for Insert Configuration>\n";
     return ERROR_IN_EVALUATION;
    }
   return INSERT_CONFIGURATION;
  }
 else if (strcmp(function,"CUD")==0)
  {
   if (arguments<3 || arguments>3)
    {
     cerr << "<ERROR: Invalid number arguments for Update Configuration Description>\n";
     return ERROR_IN_EVALUATION;
    }
   return UPDATE_CONFIGURATION_DESCRIPTION;
  }
 else if (strcmp(function,"CGM")==0)
  {
   if (arguments<2 || arguments >2)
    {
     cerr << "<ERROR: Invalid number arguments for Get Configuration Manager>\n";
     return ERROR_IN_EVALUATION;
    }
   return GET_CONFIGURATION_MANAGER;
  }
 else if (strcmp(function,"CDT")==0)
  {
   if (arguments<3 || arguments>3)
    {
     cerr << "<ERROR:  Invalid number arguments for Checkout Configuration>\n";
     return ERROR_IN_EVALUATION;
    }
   return DUMP_CONFIGURATION_OPERATORS;
  }
 else if (strcmp(function,"CRL")==0)
  {
   if (arguments<2 || arguments>2)
    {
     cerr << "<ERROR:  Invalid number arguments for Release Configuration Lock>\n";
     return ERROR_IN_EVALUATION;
    }
   return RELEASE_CONFIGURATION_LOCK;
  }
```

```
else if (strcmp(function,"CAA")==0)
  {
   if (arguments<2 || arguments>2)
     {
      cerr << "<ERROR:  Invalid number arguments for Checkin Configuration>\n";
      return ERROR_IN_EVALUATION;
     }
   return ADD_CONFIGURATION_OPERATORS;
  }
else if (strcmp(function,"CLL")==0)
  {
   if (arguments<2 || arguments>2)
     {
      cerr << "<ERROR:  Invalid number arguments for List Configuration Operators>\n";
      return ERROR_IN_EVALUATION;
     }
   return LONG_LIST_CONFIGURATION_OPERATORS;
  }
else if (strcmp(function,"CLV")==0)
  {
   if (arguments<2 || arguments>2)
     {
      cerr << "<ERROR:  Invalid number arguments for List Default Configuration Operator>\n";
      return ERROR_IN_EVALUATION;
     }
   return LIST_CONFIGURATION_DEFAULT_OPERATOR;
  }
else if (strcmp(function,"CLO")==0)
  {
   if (arguments<2 || arguments>2)
     {
      cerr << "<ERROR:  Invalid number arguments for List Configuration Operators>\n";
      return ERROR_IN_EVALUATION;
     }
   return LIST_CONFIGURATION_OPERATORS;
  }
else if (strcmp(function,"CUM")==0)
  {
   if (arguments<3 || arguments>3)
     {
      cerr << "<ERROR:  Invalid number arguments for Update Configuration Manager>\n";
      return ERROR_IN_EVALUATION;
     }
   return UPDATE_CONFIGURATION_MANAGER;
```

```
  }
else if (strcmp(function,"CDA")==0)
  {
   if (arguments<2 || arguments >2)
     {
      cerr << "<ERROR: Invalid number of  arguments for Get Configuration Date>\n";
      return ERROR_IN_EVALUATION;
     }
   return GET_CONFIGURATION_DATE;
  }
else if (strcmp(function,"CDS")==0)
  {
   if (arguments<2 || arguments >2)
     {
      cerr << "<ERROR: Invalid number arguments ...Dump Configuration Summary>\n";
      return ERROR_IN_EVALUATION;
     }
   return DUMP_CONFIGURATION_SUMMARY;
  }
else if (strcmp(function,"CDC")==0)
  {
   if (arguments<2 || arguments >2)
     {
      cerr << "<ERROR: Invalid number arguments ...Get Last Date Changed>\n";
      return ERROR_IN_EVALUATION;
     }
   return GET_CONFIGURATION_CHANGED;
  }
else if (strcmp(function,"CPL")==0)
  {
   if (arguments<3 || arguments>3)
     {
      cerr << "<ERROR: Invalid number arguments for Post to Configuration Log>\n";
      return ERROR_IN_EVALUATION;
     }
   return POST_CONFIGURATION_LOG;
  }
else if (strcmp(function,"CGL")==0)
  {
   if (arguments<2 || arguments >2)
     {
      cerr << "<ERROR: Invalid number arguments for Get Configuration Log>\n";
      return ERROR_IN_EVALUATION;
     }
```

```
    return GET_CONFIGURATION_LOG;
   }
  else if (strcmp(function,"CAO")==0)
   {
    if (!arguments == 5)
     {
      cerr << "<ERROR: Not enough arguments for Attach Operators>\n";
      return ERROR_IN_EVALUATION;
     }
    return ATTACH_OPERATOR;
   }
  else
   {
    cerr << "Error in Configuration Command syntax \n\n";
   }
};


int evaluate_vobject_function(char *function, int arguments)
{
 if (strcmp(function,"VLO")==0)
   {
    if (arguments < 2 || arguments > 4)
     {
      cerr << "<ERROR: Invalid number arguments for List Operators>\n";
      return ERROR_IN_EVALUATION;
     }
    return LIST_OPERATORS;              // tell main() to run List Operators
   }
  else if (strcmp(function,"VUD")==0)
   {
    if (!arguments == 5)
     {
      cerr << "<ERROR: Invalid number of  arguments for Update VOBJECT Description>\n";
      return ERROR_IN_EVALUATION;
     }
    return UPDATE_VOBJECT_DESCRIPTION;
   }
  else if (strcmp(function,"VGD")==0)
   {
    if (arguments < 2 || arguments > 4)
     {
      cerr << "<ERROR: Not enough arguments to Get VOBJECT Description>\n";
      return ERROR_IN_EVALUATION;
```

```
    }
   return GET_VOBJECT_DESCRIPTION;
  }

else if (strcmp(function,"VDD")==0)
  {
   if (arguments < 2 || arguments > 4)
     {
      cerr << "<ERROR: Invalid number of  arguments for Get VOBJECT Date>\n";
      return ERROR_IN_EVALUATION;
     }
   return GET_VOBJECT_DATE;
  }

else if (strcmp(function,"VGV")==0)
  {
   if (!arguments == 2)
     {
      cerr << "<ERROR: Invalid number arguments for Get VOBJECT Versions>\n";
      return ERROR_IN_EVALUATION;
     }
   return GET_VOBJECT_VERSIONS;
  }
else if (strcmp(function,"VVV")==0)
  {
   if (arguments<1 || arguments >1)
     {
      cerr << "<ERROR: Invalid number of  arguments for Get VOBJECT Current Version>\n";
      return ERROR_IN_EVALUATION;
     }
   return GET_VOBJECT_VERSION;
  }

else if (strcmp(function,"VGL")==0)
  {
   if (arguments < 2 || arguments > 4)
     {
      cerr << "<ERROR: Invalid number of  arguments for Get VOBJECT Lock>\n";
      return ERROR_IN_EVALUATION;
     }
   return GET_VOBJECT_LOCK;
  }

else if (strcmp(function,"VDA")==0)
```

```
  {
   if (arguments<2 || arguments >4)
    {
     cerr << "<ERROR: Invalid number of  arguments for Get VOBJECT Summary>\n";
     return ERROR_IN_EVALUATION;
    }
   return DUMP_VOBJECT_SUMMARY;
  }

else if (strcmp(function,"VGP")==0)
  {
   if (arguments < 2 || arguments > 4)
    {
     cerr << "<ERROR: Invalid number of  arguments for Get VOBJECT Postscript>\n";
     return ERROR_IN_EVALUATION;
    }
   return GET_VOBJECT_POSTSCRIPT;
  }
else if (strcmp(function,"VGG")==0)
  {
   if (arguments < 2 || arguments > 4 )
    {
     cerr << "<ERROR: Invalid number of  arguments for Get VOBJECT GRAPH>\n";
     return ERROR_IN_EVALUATION;
    }
   return GET_VOBJECT_GRAPH;
  }

else if (strcmp(function,"VGI")==0)
  {
   if (arguments < 2 || arguments > 4)
    {
     cerr << "<ERROR: Invalid number of  arguments for Get VOBJECT Implementation>\n";
     return ERROR_IN_EVALUATION;
    }
   return GET_VOBJECT_IMPLEMENTATION;
  }

else if (strcmp(function,"VGC")==0)
  {
   if (arguments < 2 || arguments > 4)
    {
     cerr << "<ERROR: Invalid number of  arguments for Get VOBJECT Specification>\n";
     return ERROR_IN_EVALUATION;
```

```
    }
   return GET_VOBJECT_SPECIFICATION;
  }
else if (strcmp(function,"VGS")==0)
  {
   if (arguments < 2 || arguments > 4)
    {
     cerr << "<ERROR: Invalid number of arguments for Get VOBJECT Source>\n";
     return ERROR_IN_EVALUATION;
    }
   return GET_VOBJECT_SOURCE;
  }
else if (strcmp(function,"VDS")==0)
  {
   if (arguments < 2 || arguments > 4)
    {
     cerr << "<ERROR: Invalid number of arguments for Dump VOBJECT Source>\n";
     return ERROR_IN_EVALUATION;
    }
   return DUMP_VOBJECT_SUMMARY;
  }
else if (strcmp(function,"VAA")==0)
  {
   if (arguments < 2 || arguments >4)
    {
     cerr << "<ERROR: Invalid number of arguments for Add VOBJECT Subtree>\n";
     return ERROR_IN_EVALUATION;
    }
   return ADD_NEW_VARIATION;
  }
else if (strcmp(function,"VDF")==0)
  {
   if (arguments < 3 || arguments > 5)
    {
     cerr << "<ERROR: Invalid number of arguments for Dump VOBJECT FILE(S)>\n";
     return ERROR_IN_EVALUATION;
    }
   return DUMP_VOBJECT_FILES;
  }
else if (strcmp(function,"VLL")==0)
  {
   if (arguments < 2 || arguments > 4)
    {
     cerr << "<ERROR: Invalid number of arguments for long list Operators>\n";
```

```
      return ERROR_IN_EVALUATION;
     }
    return LONG_LIST_OPERATORS;
   }
 else if (strcmp(function,"VLP")==0)
   {
    if (arguments < 2 || arguments > 4)
     {
      cerr << "<ERROR: Invalid number of  arguments for list Parent/Siblings>\n";
      return ERROR_IN_EVALUATION;
     }
    return LONG_LIST_PARENTS;
   }
 else if (strcmp(function,"VLC")==0)
   {
    if (!arguments>1)
     {
      cerr << "<ERROR: Invalid number of  arguments for list Children>\n";
      return ERROR_IN_EVALUATION;
     }
    return LONG_LIST_CHILDREN;
   }
 else if (strcmp(function,"VDT")==0)
   {
    if (arguments < 3 || arguments > 5)
     {
      cerr << "<ERROR: Invalid number of  arguments for Dump VOBJECT TREE FILE(S)>\n";
      return ERROR_IN_EVALUATION;
     }
    return DUMP_VOBJECT_TREE;
   }
 else if (strcmp(function,"VRO")==0)
   {
    if (arguments != 4)
     {
      cerr << "<ERROR: Invalid number of  arguments for Release Operator Lock>\n";
      return ERROR_IN_EVALUATION;
     }
    return RELEASE_OPERATOR_LOCK;
   }
 else if (strcmp(function,"VRS")==0)
   {
    if (arguments != 4)
     {
```

```
        cerr << "<ERROR: Invalid number of  arguments for Release Operator Subtree Locks>\n";
        return ERROR_IN_EVALUATION;
       }
     return RELEASE_SUBTREE_LOCK;
    }

  else
    {
     cerr << "<<<Error in VOBJECT Command syntax>>> \n\n";
    }

};


int  evaluate_prototype_function(char *function, int arguments)
{
 TRACER("evaluate_prototype_function");
 if (strcmp(function,"PLN")==0)
   {
    if (!arguments==0)
      {
       cerr << "<ERROR: Too many arguments for List Prototype Names>\n";
       return ERROR_IN_EVALUATION;
      }
    else
     return LIST_PROTOTYPES:              // tell main() to run List_Prototypes
   }
 else if (strcmp(function,"PLL")==0)
   {
    if (!arguments==0)
      {
       cerr << "<ERROR: Invalid number of  arguments for Long List Prototypes>\n";
       return ERROR_IN_EVALUATION;
      }
    return LONG_LIST_PROTOTYPES;
   }
 else if (strcmp(function,"PDS")==0)
   {
    if (arguments<1 || arguments >1)
      {
       cerr << "<ERROR: Invalid number of  arguments for Dump Prototype Summary>\n";
       return ERROR_IN_EVALUATION;
      }
    return DUMP_PROTOTYPE_SUMMARY;
```

```
  }
else if (strcmp(function,"PGL")==0)
  {
   if (arguments<1 || arguments >1)
     {
      cerr << "<ERROR: Invalid number of  arguments for Get Prototype Leader>\n";
      return ERROR_IN_EVALUATION;
     }
   return GET_PROTOTYPE_LEADER;
  }
else if (strcmp(function,"PGD")==0)
  {
   if (arguments<1 || arguments >1)
     {
      cerr << "<ERROR: Invalid number arguments for Get Prototype Description>\n";
      return ERROR_IN_EVALUATION;
     }
   return GET_PROTOTYPE_DESCRIPTION;
  }
else if (strcmp(function,"PRD")==0)
  {
   if (arguments<1 || arguments >1)
     {
      cerr << "<ERROR: Invalid number arguments for Retrieve Prototype Date>\n";
      return ERROR_IN_EVALUATION;
     }
   return RETRIEVE_PROTOTYPE_DATE;
  }
else if (strcmp(function,"PIP")==0)
  {
   if (!arguments>0)
     {
      cerr << "<ERROR: Invalid number arguments for Insert Prototype>\n";
      return ERROR_IN_EVALUATION;
     }
   return INSERT_PROTOTYPE;
  }
else if (strcmp(function,"PUL")==0)
  {
   if (arguments<2 || arguments >2)
     {
      cerr << "<ERROR: Invalid number arguments for Update Leader>\n";
      return ERROR_IN_EVALUATION;
     }
```

```
    return UPDATE_PROTOTYPE_LEADER;
  }
else if (strcmp(function,"PUD")==0)
  {
   if (! arguments==1)
     {
      cerr << "<ERROR: Invalid number arguments for Update Description>\n";
      return ERROR_IN_EVALUATION;
     }
   return UPDATE_PROTOTYPE_DESC;
  }
else if (strcmp(function,"PUN")==0)
  {
   if (! arguments==1)
     {
      cerr << "<ERROR: Invalid number arguments for Update Name>\n";
      return ERROR_IN_EVALUATION;
     }
   return UPDATE_PROTOTYPE_NAME;
  }

else if (strcmp(function,"PGC")==0)
  {
   if (arguments<1 || arguments>1)
     {
      cerr << "<ERROR: Invalid number arguments for Get_Latest_Configuration>\n";
      return ERROR_IN_EVALUATION;
     }
   return GET_LATEST_CONFIGURATION;
  }

else if (strcmp(function,"PLV")==0)
  {
   if (!arguments==0)
     {
      cerr << "<ERROR: Too many arguments for List Prototype Variations>\n";
      return ERROR_IN_EVALUATION;
     }
   return PROTOTYPE_VARIATION_LIST;
  }
else if (strcmp(function,"PVA")==0)
  {
   if (!arguments==0)
     {
```

```
      cerr << "<ERROR: Too many arguments for List Prototype Variations and Versions>\n";
      return ERROR_IN_EVALUATION;
     }
    return PROTOTYPE_VAR_AND_VER_LIST;
   }
 else if (strcmp(function,"PVV")==0)
   {
    if (arguments<1 || arguments >1)
     {
      cerr << arguments << "\n";
      cerr << "<ERROR: Too many arguments for List Prototype Versions of a Variation>\n";
      return ERROR_IN_EVALUATION;
     }
    return PROTOTYPE_VERSION_LIST;
   }


 else
   {
    cerr << "Error in Prototype Command syntax \n\n";
   }

};
```

```
// File Header ------------------------------------------------------------
 //.............:
 //.Filename.....: evaluation.h
// Date        : 9/16/91
// Author      : Garry Lewis
//             : Drew Dwyer
// Modified by  : Michael D. O'Loughlin
// Date        : 6/18/92
// Modifications : Extensive modifications have been conducted on almost all of the
/*              original code written by Dwyer and Lewis.  Every function that
                accesses an attribute of an object or the entire object has had
                to be modified. The original code is not even recognizable in
                some functions.
                Because the functionality of the design database system changed
                completely from what Dwyer and Lewis developed each individual
                modification has not been documented. Because of the massive
                changes required I discarded a lot of obsolete code and build a
                new system on the base Dwyer and Lewis had established.
*/
// Compiler     : Glockenspiel C++ 2.1
//
// End header comments ----------------------------------------------------



#ifndef __EVALUATION_H
#define __EVALUATION_H

 // SCCS ID follows: will compile to place date/time stamp in object file

 static char evaluation_h_SccsId[] = "@(#)evaluation.h 1.3\t9/16/91";

// Contents ---------------------------------------------------------
 //
 //   Prototypes of functions to evaluate the command line TAG
 //   argument and set the appropriate case statement in main.
 //
 //
 // End --------------------------------------------------------------

 // Interface Dependencies -------------------------------------------
 //
 // NONE
 //
```

```
// End Interface Dependencies ------------------------------------

char charupper(char c);
char *upper(char *argument);
int evaluate_historical_function(char *function, int arguments);
int evaluate_configuration_function(char *function, int arguments);
int evaluate_vobject_function(char *function, int arguments);
int evaluate_prototype_function(char *function, int arguments);

// Description ---------------------------------------------------
//
//   charupper
//
//   Converts lower case letters to upper case.
//
//   upper
//
//   Converts the command line TAG field to upper case.  Calls
//   charupper to convert each letter.
//
//   evaluate_configuration_function
//
//   Determines the appropriate case statement to be executed for
//   database operation pertaining to configurations.
//
//   evaluate_vobject_function
//
//   Determines the appropriate case statement to be executed for
//   database operations pertaining to versioned objects.
//
//   evaluate_prototype_function
//
//   Determines the appropriate case statement to be executed for
//   database operations pertaining to prototypes.
//
// End Description -----------------------------------------------

#endif // __EVALUATION_H
```

```
// File Header ------------------------------------------------------------
 //.............:
 //.Filename.....: nodesupport.h
// Date         : 9/16/91
// Author       : Garry Lewis
//              : Drew Dwyer
// Modified by  : Michael D. O'Loughlin
// Date         : 6/18/92
// Modifications : Extensive modifications have been conducted on almost all of the
/*              original code written by Dwyer and Lewis.  Every function that
                accesses an attribute of an object or the entire object has had
                to be modified. The original code is not even recognizable in
                some functions.
                Because the functionality of the design database system changed
                completely from what Dwyer and Lewis developed each individual
                modification has not been documented. Because of the massive
                changes required I discarded a lot of obsolete code and build a
                new system on the base Dwyer and Lewis had established.
*/
// Compiler     : Glockenspiel C++ 2.1
//
// End header comments ----------------------------------------------------



#ifndef __NODESUPPORT_H
#define __NODESUPPORT_H


 // SCCS ID follows: will compile to place date/time stamp in
 // object file

 static char nodesupport_h_SccsId[] = "@(#)nodesupport.h      1.3\t9/16/91";

// Contents ---------------------------------------------------
 //
 //   NODESUPPPORT HEADER FILE
 //
 // Description
 //
 //   SIMPLE NODE SUPPORT Functions
 //
 // End --------------------------------------------------------
```

```
// Interface Dependencies ------------------------------------

#ifndef __TREENODE_H
#include "treenode.h"
#endif

// End Interface Dependencies --------------------------------

TREENODE* root_find(TREENODE_linkedlist list_to_search, const char* str);
int str_suffix_check(char* str,char ch);
int proper_super_string(char* str1, char* str2);
int proper_super_NODE_check(TREENODE* node_ptr, char* target_string);

// Description ----------------------------------------------
//
//   root_find
//
//   Locates the root node associated with given operator name.
//
//   str_suffix_check
//
//   Locates the suffix of a given string.
//
//   proper_super_string
//
//   Determines whether one string is a prefix of another string
//   (i.e., one operator is the child of another operator).
//
//   proper_super_NODE_check
//
//   Determines whether a given node should be added to the childlist
//   of a given operator name.
//
// End Description ------------------------------------------

#endif  // end nodesupport header file


// File Header ----------------------------------------------------------
//.............:
//.Filename.....: nodesupport.cxx
//.SCCS ID......: 1.3
//.Release No....: 1
//.Date.........: 9/16/91
```

```
//.Author.....: Garry Lewis
//.............: Drew Dwyer
//.Compiler.....: Glockenspiel C++ 2.1
//.............:
// End header comments ------------------------------------------------

// SCCS ID follows: will compile to place date/time stamp in
// object file

static char nodesupport_cxx_SccsId[] = "@(#)nodesupport.cxx     1.3\t9/16/91";

// Contents ----------------------------------------------------
//
//   root_find
//   str_suffix_check
//   proper_super_string
//   proper_super_NODE_check
//
// End ---------------------------------------------------------

// Interface Dependencies --------------------------------------
// TURN ON TRACE-DEBUG
#define DEBUG
#include "debug.h"

extern "C--"
{
#include <string.h>
}

#ifndef __NODESUPPORT_H
#include "nodesupport.h"
#endif

// End Interface Dependencies ----------------------------------


TREENODE* root_find(TREENODE_linkedlist list_to_search, const char* str)

   // Summary ----------------------------------------------------
   //
   //   this function returns the address of the NODE in the list
   //   that has its operator_name matching *str.
   //
```

```
    // End Summary ----------------------------------------------

{
 slist_iterator ret_node(list_to_search);
 TREENODE* node;

// BELOW SHOULD BE REPLACED WITH THE ABOVE
 while (ret_node())
   {
    node = ret_node();
    if ( !strcmp(str,node->getname() ) )  return node;
   }
 return NULL;
}


int str_suffix_check(char* str,char ch)

    // Summary --------------------------------------------------
    //
    //   this function checks to see if the char at address str is ch
    //   and that ch only appears in the string *str at this position.
    //   Thus *str with ch  = '.' is of the form ".example with no more
    //   periods" .
    //
    // End Summary ----------------------------------------------

{
 if (*str != ch ) return 0;
 else  // check for "ch"   in rest of string
   {
    if (strchr(str+1, ch ) ) return 0;
    else return 1;
   }
}


int proper_super_string(char* str1, char* str2)

    // Summary --------------------------------------------------
    //
    //   This function checks to see if str1 is a candidate to be a
    //   child of str2 in the multi-way tree.
    //
```

```
    // End Summary ----------------------------------------------

{
 // check to see if str2 is a prefix of str1
 if (str1 != strstr(str1,str2) )
   return 0;
 else
   {
    char* loc = str1 + strlen(str2);
    return str_suffix_check(loc , '.');
   }
}

int proper_super_NODE_check(TREENODE* node_ptr, char* target_string)

    // Summary ----------------------------------------------------
    //
    //   This functions checks to see if the NODE returned by
    //   ListIterator should be added to the child_list of the NODE
    //   associated with target string.  It return a 1 if it should.
    //
    // End Summary ----------------------------------------------

{
 return proper_super_string(node_ptr->getname(), target_string) ;
}


// File Header ------------------------------------------------------------
  //.............:
  //.Filename.....: protfunc.h
// Date         : 9/16/91
// Author       : Garry Lewis
//              : Drew Dwyer
// Modified by  : Michael D. O'Loughlin
// Date         : 6/18/92
// Modifications : Extensive modifications have been conducted on almost all of the
/*              original code written by Dwyer and Lewis.  Every function that
                accesses an attribute of an object or the entire object has had
                to be modified. The original code is not even recognizable in
                some functions.
                Because the functionality of the design database system changed
                completely from what Dwyer and Lewis developed each individual
                modification has not been documented. Because of the massive
```

```
            changes required I discarded a lot of obsolete code and build a
            new system on the base Dwyer and Lewis had established.
*/
// Compiler     : Glockenspiel C++ 2.1
//
// End header comments ------------------------------------------------------


#ifndef __PROTFUNC_H
#define __PROTFUNC_H

 // SCCS ID follows: will compile to place date/time stamp in
 // object file

 static char protfunc_h_SccsId[] = "@(#)protfunc.h     1.3\t9/16/91";

// Contents ----------------------------------------------------------------
 //
 //   Prototypes for functions related to manipulating
 //   instances of the class PROTOTYPE.
 //
 // End ---------------------------------------------------------------------

 void list_prot_func(int);
void long_list_prot_func(int);

void list_prot_variations(int);
void list_prot_versions(int);
void list_prot_var_and_ver(int);

void get_prot_leader_func(int, char*);
void get_prot_description_func(int, char*);
void dump_prot_summary_func(int, char *);
void retrieve_prot_date_func(int, char*);
void insert_prot_func(int, char*, char*, char*);
void update_prot_leader_func(int, char*, char*);
void update_prot_desc_func(int, char*, char*);
void update_prot_name_func(int, char*, char*);

// Description -------------------------------------------------------------
 //
 //   list_prot_func
 //
 //   Provide the name of prototypes in the design database.
```

```
//
//   long_list_prot_func
//
//   Provides a list of all prototypes, the default configuration
//   assigned to a prototype and the version number of the root
//   versioned object.
//
//   get_prot_leader_func
//
//   Provides the name of the leader assigned to a prototype.
//
//   get_prot_description_func
//
//   Provides the description of a given prototype.
//
//   dump_prot_summary_func
//
//   Provides a summary of the prototype.  Include creation date,
//   leader, default configuration, and a description.
//
//   retrieve_prot_date_func
//
//   Provides the creation date.
//
//   insert_prot_func
//
//   Creates a new prototype in the database.
//
//   update_prot_leader_func
//
//   Changes the prototype leader's name.
//
//   update_prot_desc_func
//
//   Changes the description of a prototype.
//
//   update_prot_name_func
//
//   Changes the prototype name.
//
// End Description ---------------------------------------------------

#endif // __PROTFUNC_H
```

```
// File Header ------------------------------------------------------------
 //.............:
 //.Filename.....: protfunc.cxx
// Date        : 9/16/91
// Author      : Garry Lewis
//             : Drew Dwyer
// Modified by  : Michael D. O'Loughlin
// Date        : 6/18/92
// Modifications : Extensive modifications have been conducted on almost all of the
/*              original code written by Dwyer and Lewis.  Every function that
                accesses an attribute of an object or the entire object has had
                to be modified. The original code is not even recognizable in
                some functions.
                Because the functionality of the design database system changed
                completely from what Dwyer and Lewis developed each individual
                modification has not been documented. Because of the massive
                changes required I discarded a lot of obsolete code and build a
                new system on the base Dwyer and Lewis had established.
*/
// Compiler     : Glockenspiel C++ 2.1
//
// End header comments ----------------------------------------------------



// SCCS ID follows: will compile to place date/time stamp in
// object file

static char protfunc_cxx_SccsId[] = "@(#)protfunc.cxx   1.3\t9/16/91";

// Contents ---------------------------------------------------
//
//   list_prot_func
//   long_list_prot_func
//   get_prot_leader_func
//   get_prot_description_func
//   dump_prot_summary_func
//   retrieve_prot_date_func
//   insert_prot_func
//   update_prot_leader_func
//   update_prot_desc_func
//   update_prot_name_func
//
// End --------------------------------------------------------
```

```
// Implementation Dependencies --------------------------------


// TURN ON TRACE-DEBUG

#define DEBUG
#include "debug.h"

#ifndef __DDBDEFINES_H
#include "ddbdefines.h"
#endif

#include "My_String.h"
#include <stream.hxx>
#include <List.h>
#include <Directory.h>

extern "C--"
{
#include <sys/time.h>
#include <sys/types.h>
}

#ifndef __PROTOTYPE_H
#include "prototype.h"
#endif

// End Implementation Dependencies------------------------------


// Interface Dependencies -------------------------------------

#ifndef __PROTFUNC_H
#include "protfunc.h"
#endif

// End Interface Dependencies ---------------------------------

extern List *myPrototypeList;
extern char *ddbRootDir;
PROTOTYPE *prototypePtr;
ifstream inFile;
```

```
void list_prot_func(int number_arguments)
{
 switch (number_arguments)
   {
   case 0:
     {
      OC_setWorkingDirectory(ddbRootDir);
      ListIterator my_iterate(myPrototypeList);
      while(my_iterate.moreData())
        {
         cout << (char *)my_iterate() << "\n";
        }
     }
     break;
   default:
     cerr << "<ERROR: problem listing prototypes in database>\n";
   }
}

void long_list_prot_func(int number_arguments)
{
 char *proto_name = (char *)0;
 char *configname = (char *)0;
 switch (number_arguments)
   {
   case 0:
     {
      OC_setWorkingDirectory(ddbRootDir);
      List &protoReference = *myPrototypeList;
      List *proto_names = (List*)0;
      proto_names = new List(protoReference);
      ListIterator my_iterate(proto_names);
      while(my_iterate.moreData())
        {
         Directory *prototype_dir=(Directory*)0;
         char *proto_name = (char *)0;
         char *name = (char *)0;
         proto_name = (char *)my_iterate();

         name = (char *)(My_String(proto_name) + My_String("_dir") );

         prototype_dir = (Directory *)OC_lookup(name);
         if (prototype_dir)
           OC_setWorkingDirectory(prototype_dir);
```

```
char *prototype_name = (char*)(My_String(proto_name) +
  My_String(PROTOTYPE_EXT));
prototypePtr = (PROTOTYPE*)OC_lookup(prototype_name);
if (prototypePtr)
 {
  CONFIGURATION *configPtr;
  configPtr = prototypePtr->getDefaultConfiguration();
  if (configPtr)
   {
    configname =new char [strlen(configPtr->name()+1)];
    strcpy(configname,configPtr->name());
   }
  else
   {
    configname = new char [2];
    strcpy(configname," ");
   }
  cout << proto_name;
  int i=0;
  for (i=0;i<(PRINT_CONFIG_LOCATION-strlen(proto_name));i++)
   cout << " ";
  cout << configname;
  for (i=0;i<(PRINT_VERSION_LOCATION-
          (PRINT_CONFIG_LOCATION+strlen(configname)));i++)
   cout << " ";

  V_OBJECT *vobjectPtr = prototypePtr->getVobject();
  if (vobjectPtr)
   {
    int variation = vobjectPtr->getVariationNumber();
    cout << variation << " ";;
   }
  else
   cout << "";

  if (vobjectPtr)
   {
    int version = vobjectPtr->getVersionNumber();
    cout << version << "\n";;
   }
  else
   cout << "" << "\n";
  delete name;
```

```
            delete configname;
           }
         }
      }
      break;
    default:
      cerr << "<ERROR: problem long listing prototypes in database>\n";
    }
}

void list_prot_variations(int number_arguments)
{
 switch (number_arguments)
   {
   case 0:
     {
     cerr << "In list prototype variations\n";
     }
   break;
   default:
     cerr << "<ERROR: Problem listing prototype variations\n";
   }
}

void list_prot_versions(int number_arguments)
{
 switch (number_arguments)
   {
   case 1:
     {
     cerr << "In list prototype versions\n";
     }
   break;
   default:
     cerr << "<ERROR: Problem listing prototype versions\n";
   }
}
void list_prot_var_and_ver(int number_arguments)
{
 switch (number_arguments)
   {
   case 0:
     {
     cerr << "In list prototype variations and versions\n";
```

```
    }
   break;
   default:
    cerr << "<ERROR: Problem listing prototype variations and versions\n";
   }
}

void get_prot_leader_func(int number_arguments, char *arg1)
{
 switch (number_arguments)
   {
   case 1:
    char *prototype_name = new char [strlen(arg1)+5];
    strcpy(prototype_name,arg1);
    strcat(prototype_name,PROTOTYPE_EXT);
    prototypePtr = (PROTOTYPE*)OC_lookup(prototype_name);
    prototypePtr -> getPrototypeLeader();
    break;
   default:
    cerr << "<ERROR: extra arguments in get description call\n";
   }
}


void get_prot_description_func(int number_arguments, char *arg1)
{

 switch (number_arguments)
   {
   case 1:
    char *prototype_name = new char [strlen(arg1)+5];
    strcpy(prototype_name,arg1);
    strcat(prototype_name,PROTOTYPE_EXT);
    prototypePtr = (PROTOTYPE*)OC_lookup(prototype_name);
    prototypePtr -> getPrototypeDescription();
    break;
   default:
    cerr << "<ERROR: extra arguments in get description call\n";
   }
}

void dump_prot_summary_func(int number_arguments, char *arg1)
{
```

```
 switch (number_arguments)
   {
   case 1:
    char* prototype_name =
          (char*)(My_String(arg1)+My_String(PROTOTYPE_EXT));
    prototypePtr = (PROTOTYPE*)OC_lookup(prototype_name);
    prototypePtr -> dumpPrototypeSummary();
    break;
   default:
    cerr << "<ERROR: extra arguments in dump Prototype Summary call\n";
   }
}


void retrieve_prot_date_func(int number_arguments, char *arg1)
{
 switch (number_arguments)
   {
   case 1:
    char* prototype_name =
          (char*)(My_String(arg1)+My_String(PROTOTYPE_EXT));
    prototypePtr = (PROTOTYPE*)OC_lookup(prototype_name);
    if (prototypePtr)
     {
      time_t protTime = prototypePtr -> getProtCreationDate();
      cout << ctime(&protTime);
     }
    else
     {
      cerr << "<Prototype " << arg1 << " not found>\n"
        << "<find prototype creation date operation aborted>";
     }
    break;
   default:
    cerr << "<ERROR: invalid number args for prototype time retrieval>\n";
   }
}


void insert_prot_func(int number_arguments, char *arg1, char *arg2, char *arg3)
{

    char* prototype_name =
          (char*)(My_String(arg1)+My_String(PROTOTYPE_EXT));
```

```
prototypePtr = (PROTOTYPE *)OC_lookup(prototype_name);
if (!prototypePtr)
  {
   switch (number_arguments)
    {
    case 1:
     prototypePtr = new PROTOTYPE(prototype_name);
     OC_setWorkingDirectory(ddbRootDir);
     myPrototypeList -> Insert(arg1);
     myPrototypeList -> putObject();
     break;
    case 2:
    case 3:
     prototypePtr = new PROTOTYPE(prototype_name, arg2);
     inFile.open(arg3, ios::in);
     if (inFile)
      prototypePtr -> updatePrototypeDescription(arg3,inFile);
     OC_setWorkingDirectory(ddbRootDir);
     myPrototypeList -> Insert(arg1);
     myPrototypeList -> putObject();
     inFile.close();
     break;
    default:
     cerr << "<ERROR: invalid number args for insert prototype>\n";
    }
  }
 else
  {
   cerr << "<NOTE: " << arg1 << " already exists!>\n";
   return;
  }
}


void update_prot_leader_func(int number_arguments, char *arg1, char *arg2)
{
 switch (number_arguments)
  {
  case 2:
   char* prototype_name =
         (char*)(My_String(arg1)+My_String(PROTOTYPE_EXT));
   prototypePtr = (PROTOTYPE*)OC_lookup(prototype_name);
   prototypePtr -> changePrototypeLeader(arg2);
   break;
```

```
  default:
    cerr << "<ERROR: invalid number args for update Leader>\n";
  }
}


void update_prot_desc_func(int number_arguments, char *arg1, char *arg2)
{
 switch (number_arguments)
  {
  case 2:
   {
    inFile.open(arg2,ios::in);
    if (!inFile)
     {
      cerr << "File with description contents does not exist\n"
        << "Aborting prototype update_description\n";
     }
    else
     {
      char *prototype_name = new char [strlen(arg1)+5];
      strcpy(prototype_name,arg1);
      strcat(prototype_name,PROTOTYPE_EXT);
      prototypePtr = (PROTOTYPE*)OC_lookup(prototype_name);
      prototypePtr -> updatePrototypeDescription(arg2,inFile);
      inFile.close();
     }
    break;
   }
  default:
    cerr << "<ERROR: invalid number args for update description>\n";
  }
}


void update_prot_name_func(int number_arguments, char *arg1, char *arg2)
{
 switch (number_arguments)
  {
  case 2:
    char* prototype_name =
          (char*)(My_String(arg1)+My_String(PROTOTYPE_EXT));
    char* new_prototype_name =
          (char*)(My_String(arg2)+My_String(PROTOTYPE_EXT));
```

```
    prototypePtr = (PROTOTYPE*)OC_lookup(prototype_name);
    prototypePtr -> changePrototypeName(new_prototype_name);
    break;
  default:
    cerr << "<ERROR: invalid number args for update Name>\n";
  }
}
```

```
// File Header ---------------------------------------------------------
//.............:
//.Filename.....: prototype.h
// Date        : 9/16/91
// Author      : Garry Lewis
//             : Drew Dwyer
// Modified by  : Michael D. O'Loughlin
// Date        : 6/18/92
// Modifications : Extensive modifications have been conducted on almost all of the
/*              original code written by Dwyer and Lewis.  Every function that
                accesses an attribute of an object or the entire object has had
                to be modified. The original code is not even recognizable in
                some functions.
                Because the functionality of the design database system changed
                completely from what Dwyer and Lewis developed each individual
                modification has not been documented. Because of the massive
                changes required I discarded a lot of obsolete code and build a
                new system on the base Dwyer and Lewis had established.
*/
// Compiler     : Glockenspiel C++ 2.1
//
// End header comments -------------------------------------------------



#ifndef __PROTOTYPE_H
#define __PROTOTYPE_H

 // SCCS ID follows: will compile to place date/time stamp in
 // object file

 static char prototype_h_SccsId[] = "@(#)prototype.h   1.3\t9/16/91";

// Contents -----------------------------------------------------
 //
 //   PROTOTYPE
 //
 // Description
 //
 //   Defines class PROTOTYPE.
 //
 // End -----------------------------------------------------------
```

```
// Interface Dependencies ------------------------------------

#include <Object.h>
#include <List.h>
#include <Dictionary.h>
#include <Reference.h>
#include "ReferenceMacros.h"

 extern "C--"
{
#include <sys/time.h>
#include <sys/types.h>
}

#ifndef __TEXT_OBJECT_H
#include "text_object.h"
#endif

#ifndef __CONFIGURATION_H
#include "configuration.h"
#endif

#ifndef __VOBJECT_H
#include "versioned_object.h"
#endif

TypeCheckReference(ConfDictReference, Reference, Dictionary);
TypeCheckReference(TextObjectReference, Reference, TEXT_OBJECT);
TypeCheckReference(DefaultConfReference, Reference, CONFIGURATION);

// End Interface Dependencies ---------------------------------

#define DEFAULT_NAME ""

 // Class //

 class PROTOTYPE : public Object
{
 private:
 char *protleader;
 int protDictIndex;
 time_t protCreationDate;
 TextObjectReference protdescription;
 ConfDictReference prot_configuration_list; //points to a Dictionary
```

```
  DefaultConfReference prot_default_configuration;

  public:
  PROTOTYPE(APL *);
  PROTOTYPE(char *prototype_name,
        char *prototype_leader=DEFAULT_NAME);
  virtual void Destroy(Boolean aborted=FALSE);
  virtual Type *getDirectType();
  char *getName();
  char *getConfigName();
  void getPrototypeName();
  void getPrototypeLeader();
  void getPrototypeDescription();
  void changePrototypeName(char *new_prototype_name);
  void changePrototypeLeader(char *new_prototype_leader);
  void updatePrototypeDescription(char*, ifstream&);
  void dumpPrototypeSummary();
  void addConfiguration(CONFIGURATION *configuration);
  void listConfigurations();
  time_t setProtCreationDate();
  time_t getProtCreationDate();
  void getDefaultConfigName();
  CONFIGURATION *getConfiguration(char *);
  CONFIGURATION *getDefaultConfiguration();
  V_OBJECT *getVobject();
  ~PROTOTYPE() { Destroy(FALSE); }

};

// Description ----------------------------------------------------
  //
  //   Defines a PROTOTYPE class.
  //
  // Constructor
  //
  //   prototype --APL
  //
  //   ONTOS required constructor
  //
  //   prototype
  //
  //   Constructs a prototype object from the given name,
  //   and optional team leader of the prototype.
  //
```

```
// Public Members
//
//   Destroy
//
//   used to cleanup memory during deletion and aborts.
//
//   getDirectType
//
//   Return the ONTOS Type of class PROTOTYPE.
//
//   getName
//
//   Returns a character string containing the name of the prototype
//
//   getConfigName
//
//   Returns a character string containing the last configuration
//   worked on by a user
//
//   getPrototypeName
//
//   Prints the prototype name to standard output
//
//   getPrototypeLeader
//
//   Displays the prototype designer team leader's name.
//
//   getPrototypeProtdescription
//
//   Displays a protdescription of the prototype.
//
//   changePrototypeName
//
//   Change the name of the prototype.
//
//   changePrototypeLeader
//
//   Change the prototype leader's name.
//
//   updatePrototypeProtdescription
//
//   Adds a protdescription to a PROTOTYPE object.
//
//   dumpPrototypeSummary
```

```
//
//  Provides date created, leader, default configuration,
//  and description of a prototype.
//
//  addConfiguration
//
//  Adds a configuration with a given name to the prototype
//
//  listConfigurations
//
//  List the names of all the configurations in the prototype.
//
//  getDefaultConfigName
//
//  Displays the name of the default configuration.
//
//  getConfiguration
//
//  Used by this class as a support function to update the default
//  configuration.
//
//  getDefaultConfiguration
//
//  returns a pointer to the default Configuration
//
//  getVobject
//
//  returns the most current root V_OBJECT (Operator or Type)
//  associated with a prototype.
//
//  ~prototype
//
//   The class destructor.
//
// End -----------------------------------------------------------

#endif // __PROTOTYPE_H
```

```
// File Header ------------------------------------------------------------
 //.............:
 //.Filename.....: prototype.cxx
// Date        : 9/16/91
// Author      : Garry Lewis
//             : Drew Dwyer
// Modified by  : Michael D. O'Loughlin
// Date        : 6/18/92
// Modifications : Extensive modifications have been conducted on almost all of the
/*              original code written by Dwyer and Lewis.  Every function that
                accesses an attribute of an object or the entire object has had
                to be modified. The original code is not even recognizable in
                some functions.
                Because the functionality of the design database system changed
                completely from what Dwyer and Lewis developed each individual
                modification has not been documented. Because of the massive
                changes required I discarded a lot of obsolete code and build a
                new system on the base Dwyer and Lewis had established.
*/
// Compiler     : Glockenspiel C++ 2.1
//
// End header comments ----------------------------------------------------



// SCCS ID follows: will compile to place date/time stamp in
// object file

static char prototype_cxx_SccsId[] = "@(#)prototype.cxx 1.3\t9/16/91";

// Contents ---------------------------------------------------
//
//   PROTOTYPE::PROTOTYPE          ONTOS constructor
//   PROTOTYPE::PROTOTYPE          constructor
//   PROTOTYPE::Destroy
//   PROTOTYPE::getDirectType
//   PROTOTYPE::getName
//   PROTOTYPE::getConfigName
//   PROTOTYPE::getPrototypeName
//   PROTOTYPE::getPrototypeLeader
//   PROTOTYPE::getPrototypeDescription
//   PROTOTYPE::getNumberOfVariations;
//   PROTOTYPE::displayNumberOfVariations;
//   PROTOTYPE::incrementVariations;
```

```
//   PROTOTYPE::changePrototypeName
//   PROTOTYPE::changePrototypeLeader
//   PROTOTYPE::updatePrototypeDescription
//   PROTOTYPE::dumpPrototypeSummary
//   PROTOTYPE::addConfiguration
//   PROTOTYPE::listConfigurations
//   PROTOTYPE::setProtCreationDate
//   PROTOTYPE::getProtCreationDate
//   PROTOTYPE::getDefaultConfigName
//   PROTOTYPE::getConfiguration
//   PROTOTYPE::getDefaultConfiguration
//   PROTOTYPE::getVobject
//
// Description
//
//   Implementation of class PROTOTYPE member functions.
//
// End -----------------------------------------------------

// Interface Dependencies ----------------------------------

// TURN ON TRACE-DEBUG
#define DEBUG
#include "debug.h"


#include <GlobalEntities.h>
#include <Directory.h>
#include <stream.hxx>

extern "C--"
{
#include <sys/time.h>
#include <strings.h>
}

#ifndef __PROTOTYPE_H
#include "prototype.h"
#endif


// End Interface Dependencies ------------------------------

extern Type *PROTOTYPE_OType;
```

```
extern Type *CONFIGURATION_OType;
extern Type *V_OBJECT_OType;

// ONTOS required constructor //

PROTOTYPE::PROTOTYPE(APL *theAPL) : (theAPL)
{
};

// New Instance Constructor //

PROTOTYPE::PROTOTYPE(char *prototype_name,
                     char *prototype_leader) : (prototype_name)

   // Summary ------------------------------------------------------------
   //
   //   Constructs a persistent prototype object.  A PROTOTYPE object
   //   contains general management information about a prototype and
   //   a reference to a container holding configuration objects.
   //
   // Parameter
   //
   //   prototype_name
   //
   //   A pointer to a character string.
   //
   //   prototype_leader
   //
   //   A pointer to a character string.
   //
   // Functional Description
   //
   //   Passes the object name to class Object.  Copies the leader's name
   //   into private data members.  We initialize the description to null
   //   and create a dictionary to hold various configurations.
   //
   // End ----------------------------------------------------------------

{
 initDirectType(PROTOTYPE_OType);

 protleader = new char[strlen(prototype_leader)+1];
 strcpy(protleader, prototype_leader);
 protDictIndex = 0;
```

```
 protCreationDate = setProtCreationDate();
 protdescription.initToNull();
 prot_default_configuration.initToNull();

 if(!CONFIGURATION_OType)
   {
    CONFIGURATION_OType = (Type*)OC_lookup("CONFIGURATION");
   }

 Dictionary *new_configuration = new Dictionary(OC_integer,
                                 CONFIGURATION_OType,
                                 TRUE, FALSE);
 new_configuration ->putObject();
 prot_configuration_list.Reset(new_configuration, this);

 putObject();
}

// End Constructor PROTOTYPE::PROTOTYPE //

//  Member Function //

void PROTOTYPE::Destroy(Boolean aborted)
{
 delete protleader;
 if (aborted)
   {
    Object::Destroy(aborted);
   }
}

Type* PROTOTYPE::getDirectType()

    // Summary ---------------------------------------------------
    //
    //   returns the ONTOS Type for the prototype class.
    //
    // Return value
    //
    //   A pointer to an ONTOS Type.
    //
    // End -------------------------------------------------------

{
```

```
 return PROTOTYPE_OType;
}

char * PROTOTYPE::getName()

    // Summary ---------------------------------------------------
    //
    //   returns the name of the prototype
    //
    // Return value
    //
    //   A pointer to a character string
    //
    // End -------------------------------------------------------

{
 Directory *directory=(Directory *)0;
 char *name=(char *)0;
 char *temp=(char *)0;

 if(!this)
   {
    cerr << "<ERROR: cannot get the name of a null PROTOTYPE>\n";
    return NULL;
   }
 else
   {
    name = Name();
    OC_getNameComponents(name, &directory, &name);
    temp = new char [strlen(name)+1];
    temp = strtok(name,".");
    return temp;
   }
}

void PROTOTYPE::getPrototypeName()

    // Summary ---------------------------------------------------
    //
    //   Displays the name of the prototype
    //
    // Return value
    //
    //   Displays the Prototype Name and a linefeed to the stdout
```

```
    //
    // End ---------------------------------------------------

{
 Directory *directory;
 char *name;

 if(!this)
   {
    cerr << "<ERROR: cannot get the name of a null PROTOTYPE>\n";
    return;
   }
 else
   {
    name = Name();
    OC_getNameComponents(name, &directory, &name);
    cout << name << "\n";
   }
}

void PROTOTYPE::getPrototypeLeader()

    // Summary -----------------------------------------------------
    //
    //   Displays the name of the prototype leader
    //
    // Return value
    //
    //   Displays the Prototype leader and a linefeed to the stdout
    //
    // End ---------------------------------------------------------


{
 if(!this)
   {
    cerr << "<ERROR: cannot get the leader's name of a null PROTOTYPE>\n";
    return;
   }
 cout <<  protleader << "\n";
}

void PROTOTYPE::getPrototypeDescription()
```

```
    // Summary ----------------------------------------------------
    //
    //   Displays the prototype description
    //
    // Return value
    //
    //    Display Description to stdout if description exists
    //
    // End --------------------------------------------------------

{
 if(!this)
   {
    cerr << "<ERROR: cannot get the description of a null PROTOTYPE>\n";
    return;
   }
 if(!protdescription)
   {
    cerr << "<ERROR: This prototype does not contain a description>\n";
    return;
   }
 else
   {
    TEXT_OBJECT* myTextObjPtr = (TEXT_OBJECT*) protdescription.Binding(this);
    myTextObjPtr -> text(cout);
    cout << "\n";
   }
}

void PROTOTYPE::changePrototypeName(char *new_prototype_name)

    // Summary ----------------------------------------------------
    //
    //    Changes the prototype name
    //
    // Parameter
    //
    // new_prototype_name
    //
    //    a character string pointer containing the new name
    //
    // Return value
    //
    //    N/A
```

```
    //
    // End ----------------------------------------------------------


{
 if(!this)
   {
    cerr << "<ERROR: cannot set the name of a null PROTOTYPE>\n";
    return;
   }
 Name(new_prototype_name);
}

void PROTOTYPE::changePrototypeLeader(char *new_prototype_leader)

    // Summary ------------------------------------------------------
    //
    //   changes the prototype leader
    //
    // Parameter
    //
    // new_prototype_leader
    //
    //   a character string pointer containing the new leader's name
    //
    // Return value
    //
    //   N/A
    //
    // End ----------------------------------------------------------

{
 if(!this)
   {
    cerr << "<ERROR: cannot change the leader of a null PROTOTYPE>\n";
    return;
   }
 delete protleader;
 protleader = new char[strlen(new_prototype_leader)+1];
 strcpy(protleader, new_prototype_leader);
 putObject();
}

void PROTOTYPE::updatePrototypeDescription(char *fileName, ifstream& input_file_stream)
```

```
    // Summary ----------------------------------------------------
    //
    //   changes the prototype description
    //
    // Parameter
    //
    //  fileName
    //
    //   a character string pointer containing the new name of
    //   the file containing the new description.
    //
    //  input_file_stream
    //
    //   the file handle of the input description file.
    //
    // Return value
    //
    //   N/A
    //
    // End ----------------------------------------------------------

{
 if (!protdescription)
   {
    TEXT_OBJECT *textObjectPtr = new TEXT_OBJECT();
    textObjectPtr -> append(fileName, input_file_stream);
    protdescription.Reset(textObjectPtr, this);
    putObject();
   }
 else
   {
    TEXT_OBJECT *textObjectPtr = (TEXT_OBJECT*) protdescription.Binding(this);
    textObjectPtr -> resetTheText();
    textObjectPtr -> append(fileName, input_file_stream);
    putObject();
   }
}

void PROTOTYPE::dumpPrototypeSummary()
{

 // Summary ----------------------------------------------------
 //
```

```
  //   Displays the date created, leader, default config, and
  //   description of a prototype to stdout
  //   1 item per line ending with the (potentially) multi-line
  //   description.
  //
  // Parameter
  //
  //   N/A
  //
  // Return value
  //
  //   N/A
  //
  // End ----------------------------------------------------------
  TRACER("dumpPrototypeSummary");
  time_t creationdate = 0;
  creationdate = getProtCreationDate();
  TRACE("Creation Date: ");
  cout << ctime(&creationdate);
  getPrototypeLeader();
  getDefaultConfigName();
  getPrototypeDescription();
}

void PROTOTYPE::addConfiguration(CONFIGURATION *configuration)
{

  // Summary ------------------------------------------------------
  //
  //  adds a configuration to the prototype
  //
  // Parameter
  //
  //  configuration
  //
  //   a pointer to the configuration to be added to the
  //   prototype.
  //
  //
  // Return value
  //
  //   N/A
  //
  // End ----------------------------------------------------------
```

```
 if(!this)
  {
   cerr << "<ERROR: cannot set the description of a null PROTOTYPE>\n";
   return;
  }
 if(!configuration)
  {
   cerr << "<ERROR: cannot give to a PROTOTYPE a null configuration>\n";
   return;
  }
 else
  {
   protDictIndex = protDictIndex + 1;
   Dictionary*confDictionaryPtr=
    (Dictionary*)prot_configuration_list.Binding(this);
   confDictionaryPtr -> Insert(protDictIndex, (Entity *)configuration);
   confDictionaryPtr -> putObject();
   prot_default_configuration.Reset(configuration, this);
   putObject();
  }
}

void PROTOTYPE::listConfigurations()

   // Summary ----------------------------------------------------
   //
   //   Display the configuration names contained in this prototype
   //   to stdout. 1 configuration / line.
   //
   // Parameter
   //
   //   N/A
   //
   // Return value
   //
   //   N/A
   //
   // End --------------------------------------------------------

{
 CONFIGURATION *the_configuration;
 char *name;
 Directory *directory;
```

```
 if(!prot_configuration_list)
   {
    cerr << "<ERROR: cannot return a prototype from an empty list>\n";
    return;
   }

 Dictionary *confDictionaryPtr =
  (Dictionary*)prot_configuration_list.Binding(this);
 DictionaryIterator configlist_iterator(confDictionaryPtr);

 while(the_configuration =(CONFIGURATION*)(Entity*)configlist_iterator())
   {
    if (name = the_configuration->Name());
    {
     OC_getNameComponents(name, &directory, &name);
     cout << name << "\n";
    }
   }
}

//Member Function //

time_t PROTOTYPE::setProtCreationDate()

   // Summary ---------------------------------------------------
   //
   // sets the creation date to system date at time of this call.
   //
   // Parameter
   //
   //   N/A
   //
   // Return value
   //
   //   time_t as a default long value containing the system time
   //   This function as a byproduct updates the protCreationDate
   //   attribute field.
   //
   // End -------------------------------------------------------

 {
  time_t mytloc=0;
```

```
 time_t theTime;
 return theTime = time(mytloc);
}

// End -----------------------------------------------------


// Member Function //

time_t PROTOTYPE::getProtCreationDate()

    // Summary -----------------------------------------------------
    //
    // Returns  the prototype's creation date
    //
    // Parameter
    //
    //    N/A
    //
    // Return value
    //
    //    time_t as a long value containing the system time
    //
    // End -----------------------------------------------------

{
 return protCreationDate;
}

// End -----------------------------------------------------


void PROTOTYPE::getDefaultConfigName()

    // Summary -----------------------------------------------------
    //
    //    Displays the default Configuration name for this prototype
    //    to stdout
    //
    // Parameter
    //
    //    N/A
    //
    // Return value
```

```
    //
    //   N/A
    //
    // End ---------------------------------------------------

{
 Directory *directory;
 char *name;

 if(!prot_default_configuration)
   {
    cerr << "\n<ERROR: No configurations are contained in this prototype.>\n\n";
   }
 else
   {
    CONFIGURATION *the_configuration =
     (CONFIGURATION*)prot_default_configuration.Binding(this);
    name = the_configuration -> Name();
    OC_getNameComponents(name, &directory, &name);
    cout << name << "\n";
   }
}

char * PROTOTYPE::getConfigName()
{

 // Summary ----------------------------------------------------
 //
 //   Returns a character string pointer to an area in memory
 //   containing the name of the default Configuration name
 //   for this prototype.
 //
 // Parameter
 //
 //   N/A
 //
 // Return value
 //
 //   character string pointer
 //
 // End ---------------------------------------------------------

 Directory *directory;
 char *name;
```

```
 if(!prot_default_configuration)
   {
    cerr << "<ERROR: No configurations are contained in this prototype.>\n\n";
    return NULL;
   }
 else
   {
    CONFIGURATION *the_configuration =
     (CONFIGURATION*)prot_default_configuration.Binding(this);
    name = the_configuration -> Name();
    OC_getNameComponents(name, &directory, &name);
    return name;
   }
}

CONFIGURATION *PROTOTYPE::getConfiguration(char *confName)

   // Summary ---------------------------------------------------
   //
   //   Used by this class as a support function to update the default
   //   configuration.
   //
   // Parameter
   //
   //   confName
   //
   //     configuration name to lookup the default configuration
   //     for this prototype
   //
   // Return value
   //
   //   Configuration pointer if successful.  Null pointer if
   //   failed.
   //
   // End -------------------------------------------------------

{
 CONFIGURATION *myConfPtr = (CONFIGURATION *)OC_lookup(confName);
 if (myConfPtr)
   {
    return myConfPtr;
   }
 else
```

```
  {
   return (CONFIGURATION*)0;
  }
}

CONFIGURATION *PROTOTYPE::getDefaultConfiguration()

  // Summary ---------------------------------------------------
  //
  //  Returns the default configuration for this prototype.
  //  The last configuration worked on.
  //
  // Parameter
  //
  //  N/A
  //
  // Return value
  //
  //  Configuration pointer if successful.  Null pointer if
  //  failed.
  //
  // End -------------------------------------------------------

{
 CONFIGURATION *the_configuration =(CONFIGURATION *)0;

 if(!prot_default_configuration)
  {
   return NULL;
  }
 else
  {
   the_configuration =
    (CONFIGURATION*)prot_default_configuration.Binding(this);          return the_configuration;
  }
}

V_OBJECT *PROTOTYPE::getVobject()

  // Summary ---------------------------------------------------
  //
  //  This method assumes that the root versioned object (V_OBJECT)
  //  has the same name as the prototype.  If this is not the case
  //  then this function will not work.  Taking the prototype name,
```

```
    //   a thread lookup is performed and the most current V_OBJECT
    //   in the thread is returned.
    //
    // Parameter
    //
    //   N/A
    //
    // Return value
    //
    //   V_OBJECT pointer if successful.  Null pointer if
    //   failed.
    //
    // End ----------------------------------------------------------

{
 THREAD *threadPtr=(THREAD*)0;
 char *name = new char [strlen(getName())+1];
 strcpy(name,getName());
 threadPtr = (THREAD *)OC_lookup(name);
 if (threadPtr)
   {
    V_OBJECT *vobjectPtr = threadPtr->current();
    if (vobjectPtr)
      return vobjectPtr;
    else
      {
       return NULL;
      }
   }
 else
   {
    return NULL;
   }
}

// end functions
```

```
// File Header ------------------------------------------------------------
 //.............:
 //.Filename.....: queue.h
// Date        : 9/16/91
// Author      : Garry Lewis
//             : Drew Dwyer
// Modified by  : Michael D. O'Loughlin
// Date        : 6/18/92
// Modifications : Extensive modifications have been conducted on almost all of the
/*              original code written by Dwyer and Lewis.  Every function that
                accesses an attribute of an object or the entire object has had
                to be modified. The original code is not even recognizable in
                some functions.
                Because the functionality of the design database system changed
                completely from what Dwyer and Lewis developed each individual
                modification has not been documented. Because of the massive
                changes required I discarded a lot of obsolete code and build a
                new system on the base Dwyer and Lewis had established.
*/
// Compiler     : Glockenspiel C++ 2.1
//
// End header comments ----------------------------------------------------



#ifndef __QUEUE_H
#define __QUEUE_H

 // SCCS ID follows: will compile to place date/time stamp in
 // object file

 static char QUEUE_h_SccsId[] = "@(#)queue.h   1.3\t9/16/91";

// Contents ---------------------------------------------------
 //
 //   QUEUE
 //
 // Description
 //
 //   IMPLEMENTS class QUEUE CONSTRUCTORS.
 //
 // End --------------------------------------------------------

 // Interface Dependencies -------------------------------------
```

```
 //

#include <iostream.hxx>

 extern "C--"{
#include <stdio.h>
#include <string.h>
 }

//
 // End Interface Dependencies ---------------------------------

 // Description ----------------------------------------------
 //
 //   Defines the slink, slist, slist_iterator,
 //            treenode_linkedlist, and treenode_queue classes
 //
 //
 // There are no ontos mechanisms in these classes.  The primary
 // purpose of these classes is to provide the support structures
 // of linked_lists and queues in order to process the CAPS
 // PROTOTYPE subdirectory.
 //
 // As a subdirectory is read, each file is analzed to determine
 // whether it is an operator/type.  If found to belong to
 // an operator which might version, the operator and operator
 // information is placed into a multi-way tree parrallelling the
 // decomposition of an operator in the CAPS system.  The
 // TREENODES are then compared against operator structures in
 // the Ontos Database in other programs documented elsewhere.
 //
 // These structures are simple in nature and can be found in
 // any good C++ textbook.  These particular examples came from
 // Bjarne Stroustrup's C++ Programming Language textbook (pg 203).
// Please refer to the textbook for further explanation of the
// data structures and how they are manipulated.
//
// End -------------------------------------------------------

class TREENODE;
class slist;
class slist_iterator;

class slink {
```

```
  friend class slist;
  friend class slist_iterator;

private:
  slink* next;
  TREENODE * e;
  slink(TREENODE * a, slink* p);
};


class slist {
  friend class slist_iterator;

private:
  slink* last;                    // last-> next is head of list

public:
   slist();                      //  { last = NULL; }
   slist(TREENODE * a);
   int insert(TREENODE * a);         // add at head of list
   int append(TREENODE * a);         // add at tail of list
   TREENODE * get();                 // return and remove head of list
   void clear();                     // remove all links
   int empty();                      // returns 1 if list is empty
   ~slist();                        //    { clear(); }
};

class slist_iterator{

private:
  slink* ce;
  slist* cs;

public:
  slist_iterator(slist& s);
  TREENODE * operator()();
};

class TREENODE_linkedlist : public slist
{
 public:
 TREENODE_linkedlist();
 int member(char *name);
};
```

```
class TREENODE_queue : private slist
{
 public:
 TREENODE_queue(){}
 void put(TREENODE * a) {append(a); }
 slist::empty;
 slist::get;
};

#endif // QUEUE Class header
```

```
// File Header ------------------------------------------------------------
 //.............:
 //.Filename.....: queue.cxx
 //.SCCS ID......: 1.3
 //.Release No....: 1
 //.Date.........: 9/16/91
 //.Author.......: Garry Lewis
 //.............: Drew Dwyer
 //.Compiler.....: Glockenspiel C++ 2.1
 //.............:
 // End header comments ---------------------------------------------------

// SCCS ID follows: will compile to place date/time stamp in
// object file

static char QUEUE_cxx_SccsId[] = "@(#)queue.cxx 1.3\t9/16/91";

// Contents --------------------------------------------------
//
//   QUEUE
//
// Description
//
//   IMPLEMENTS class QUEUE CONSTRUCTORS.
//
// End -------------------------------------------------------

// Interface Dependencies -------------------------------------
//

// TURN ON TRACE-DEBUG
#define DEBUG
#include "debug.h"


#ifndef __QUEUE_H
#include "queue.h"
#endif
//
// End Interface Dependencies ---------------------------------

//----------X-----------X-------------X-----------X----------------X
// these are the implementation methods of classes
// slink, slist, slist_iterator, and TREENODE_linkedlist
```

```
//----------X-----------X-------------X-----------X---------------X

//----------X-----------X-------------X-----------X---------------X
//
// slink methods
//
//----------X-----------X-------------X-----------X---------------X

slink::slink(TREENODE * a, slink* p)

{
 e = a;
 next = p;
}

//----------X-----------X-------------X-----------X---------------X
//
// slist methods
//
//----------X-----------X-------------X-----------X---------------X

slist::slist()

{
 last = NULL;
}

slist::slist(TREENODE * a)

{
 last = new slink(a,NULL);
 last -> next = last;
}

int slist::insert(TREENODE * a)
{
 if (last)
   last->next = new slink(a, last->next);
 else {
   last = new slink(a,NULL);
   last -> next = last;
 }
 return 0;
}
```

```
int slist::append(TREENODE * a)
{
 if (last)
   last = last -> next = new slink(a, last->next);
 else
   {
    last = new slink(a,NULL);
    last -> next = last;
   }
 return 0;
}

TREENODE * slist::get()
{
 // improve the following line for better error detection
 if (last == NULL) cout << "get from empty slist\n";
 slink* f = last-> next;
 TREENODE * r = f->e;
 if (f == last) last = NULL;
 else last ->next = f->next;
 delete f;
 return r;
}

void slist::clear()
{
 slink* l = last;
 if ( l == NULL ) return;
 do
   {
    slink* ll = l;
    l = l-> next;
    delete ll;
   } while (l != last);
}

int slist::empty()

{
 if (last == NULL) return 1;
 else return 0;
}
```

```
slist::~slist()

{
 clear();
}


//----------X-----------X-------------X-----------X---------------X
//
// slist iterator methods
//
//----------X-----------X-------------X-----------X---------------X

slist_iterator::slist_iterator(slist& s)

{
 cs = &s;
 ce = cs -> last;
}

TREENODE *slist_iterator::operator()()
{
 TREENODE * ret = ce ? (ce = ce -> next) -> e : NULL;
 if ( ce == cs ->last) ce = NULL;
 return ret;
}

//----------X-----------X-------------X-----------X---------------X
//
// TREENODE_linkedlist methods
//
//----------X-----------X-------------X-----------X---------------X

TREENODE_linkedlist::TREENODE_linkedlist()
{
}

int TREENODE_linkedlist::member(char *name)

{
 char *temp = name;
 return 1;
}
```

```
// File Header ------------------------------------------------------------
 //.............:
 //.Filename.....: text_object.h
// Date        : 9/16/91
// Author      : Garry Lewis
//             : Drew Dwyer
// Modified by  : Michael D. O'Loughlin
// Date        : 6/18/92
// Modifications : Extensive modifications have been conducted on almost all of the
/*              original code written by Dwyer and Lewis.  Every function that
               accesses an attribute of an object or the entire object has had
               to be modified. The original code is not even recognizable in
               some functions.
               Because the functionality of the design database system changed
               completely from what Dwyer and Lewis developed each individual
               modification has not been documented. Because of the massive
               changes required I discarded a lot of obsolete code and build a
               new system on the base Dwyer and Lewis had established.
*/
// Compiler     : Glockenspiel C++ 2.1
//
// End header comments ----------------------------------------------------



#ifndef __TEXT_OBJECT_H
#define __TEXT_OBJECT_H

 // SCCS ID follows: will compile to place date/time stamp in object file

 static char text_object_h_SccsId[] = "@(#)text_object.h      1.3\t9/16/91";

// Contents ---------------------------------------------------------
 //
 //   TEXT_OBJECT
 //
 // Description
 //
 //   Defines class TEXT_OBJECT.
 //
 // End --------------------------------------------------------------

 // Interface Dependencies -------------------------------------
 // SCCS ID follows: will compile to place date/time stamp in object file
```

```
 // Interface Dependencies ------------------------------------

#include <Object.h>
#include <stream.hxx>

 // End Interface Dependencies --------------------------------


 class TEXT_OBJECT : public Object
{
 private:
 char *the_file_name;
 char *the_text;
 public:
 TEXT_OBJECT(APL *);
 TEXT_OBJECT();
 void Destroy(Boolean aborted=FALSE);
 Type *getDirectType();
 void append(char *, ifstream&);
 void append(char *);
 void append(ifstream&);
 void text(ostream&); //  standard output
  Boolean rebuildTextFile(char*);
 void displayFileName();
 char *getFileName();
 char *text();
 void resetTheText();
 ~TEXT_OBJECT() { Destroy(FALSE); }
};

// Description ----------------------------------------------
 //
 //   Defines a TEXT_OBJECT class.  The class TEXT_OBJECT is a derived
 //   class of Object.
 //
 // Constructor
 //
 // TEXT_OBJECT -- APL
 //
 // ONTOS required constructor
 //
 // TEXT_OBJECT
```

```
//
// Creates a new instance of TEXT_OBJECT
//
// Public Members
//
// Destroy
//
// ONTOS required method.
//
// getDirectType
//
// ONTOS rquired method used to return the class Type.
//
// append
//
// Reads in a text file.
//
// append
//
// Read in a string append to existing string in log fashion.
//
// text
//
// Send the text contents to standard output.
//
// rebuildTextFile
//
// Write the contents of a text object to a file.
//
// displayFileName
//
// Send the file name to standard output.
//
// getFileName
//
// Return a pointer the file name contained in the text object.
//
// text
//
// Returns a pointer to the text contained in the text object.
//
// resetTheText
//
// Set the text field to a empty string.
```

```
//
// ~TEXT_OBJECT
//
// class destructor.
//
// End Description ----------------------------------------------------

#endif //__TEXT_OBJECT_H
```

```
// File Header ------------------------------------------------------------
 //.............:
 //.Filename.....: text_object.cxx
// Date        : 9/16/91
// Author      : Garry Lewis
//             : Drew Dwyer
// Modified by  : Michael D. O'Loughlin
// Date        : 6/18/92
// Modifications : Extensive modifications have been conducted on almost all of the
/*              original code written by Dwyer and Lewis.  Every function that
                accesses an attribute of an object or the entire object has had
                to be modified. The original code is not even recognizable in
                some functions.
                Because the functionality of the design database system changed
                completely from what Dwyer and Lewis developed each individual
                modification has not been documented. Because of the massive
                changes required I discarded a lot of obsolete code and build a
                new system on the base Dwyer and Lewis had established.
*/
// Compiler     : Glockenspiel C++ 2.1
//
// End header comments ----------------------------------------------------




// SCCS ID follows: will compile to place date/time stamp
// in object file

static char composite_cxx_SccsId[] = "@(#)text_object.cxx      1.3\t9/16/91";

// Contents ---------------------------------------------------
//
//   TEXT_OBJECT::TEXT_OBJECT            Ontos Constructor
//   TEXT_OBJECT::TEXT_OBJECT
//   TEXT_OBJECT::Destroy
//   TEXT_OBJECT::getDirectType
//   TEXT_OBJECT::append
//   TEXT_OBJECT::append
//   TEXT_OBJECT::append
//   TEXT_OBJECT::text
//   TEXT_OBJECT::rebuildTextFile
//   TEXT_OBJECT::displayFileName
//   TEXT_OBJECT::getFileName
```

```
//   TEXT_OBJECT::text
//   TEXT_OBJECT::resetTheText
//
// Description
//
//   Implementation of class TEXT_OBJECT member functions.
//
// End -----------------------------------------------------

// Interface Dependencies ----------------------------------

// TURN ON TRACE-DEBUG
#define DEBUG
#include "debug.h"

#include "My_String.h"

#include <strstream.hxx>
#include <stream.hxx>
#include <fstream.hxx>
#ifndef __TEXT_OBJECT_H
#include "text_object.h"
#endif

#ifndef __TRACER_H
#include "tracer.h"
#endif

#ifndef __DDBDEFINES_H
#include "ddbdefines.h"
#endif

extern "C--"
{
#include <stdio.h>
#include <sys/time.h>
#include <sys/types.h>
#include <stdlib.h>
#include <strings.h>
}
// End -----------------------------------------------------

extern Type *TEXT_OBJECT_OType;
extern char *dirNamePtr;
```

```
TEXT_OBJECT::TEXT_OBJECT(APL *theAPL) : Object(theAPL)

   // Summary ---------------------------------------------------
   //
   //   Ontos Required Constructor
   //
   // Return value
   //
   //   A TEXT_OBJECT object
   //
   // End -------------------------------------------------------

{
};

TEXT_OBJECT::TEXT_OBJECT()

   // Summary ---------------------------------------------------
   //
   //   Constructor
   //
   // Return value
   //
   //   A TEXT_OBJECT object
   //
   //   the_file_name and the_text attributes are initialized to NULL;
   // End -------------------------------------------------------

{

 the_text = (char *)My_String("");
 the_file_name = (char *)My_String("");
 putObject();
};

void TEXT_OBJECT::Destroy(Boolean aborted)

   // Summary ---------------------------------------------------
   //
   //   Constructor
   //
   // Return value
   //
```

```
    //    A TEXT_OBJECT object
    //
    //    the_file_name and the_text attributes are initialized to NULL;
    // End -----------------------------------------------------
{
 if(the_file_name);
 delete the_file_name;
 delete the_text;
 if(aborted)
   {
    Object::Destroy(aborted);
   }
};

Type *TEXT_OBJECT::getDirectType()

    // Summary -------------------------------------------------
    //
    //    Ontos required method which returns the type of this object.
    //
    // Parameter
    //
    //    N/A
    //
    // Return Value
    //
    //    the type of this object.  TEXT_OBJECT.
    //
    // End -----------------------------------------------------

{
 return TEXT_OBJECT_OType;
};

void TEXT_OBJECT::append(char *filename, ifstream& input_file)
    // Summary -------------------------------------------------
    //
    //    append a file as a text object
    //
    // Parameter
    //
    //    filename
    //
    //    character string * containing the name of the file which
```

```
    //   is read into a text_object.
    //
    //   input_file
    //
    //   file handle of input file
    //
    // Return Value
    //
    //   stores the file as a text_object in the database
    // End ----------------------------------------------------

{
TRACER("TEXT_OBJECT::append(2 params)");

 the_file_name = (char *)(My_String(filename));
 TRACE(the_file_name);

 ostrstream buf;
 char ch;
 while (buf && input_file.get(ch))
  {
   buf.put(ch);
  }
 buf.put('\0');
 the_text = buf.str();
 putObject();
};

void TEXT_OBJECT::append(char *instring)

    // Summary ------------------------------------------------
    //
    //   append a character string as a text object. stores the character
    //   string as a text_object in the database
    //
    // Parameter
    //
    //   instring
    //
    //   character string * containing the description to be added as
    //   a text_object.
    //
    // Return Value
    //
```

```
    //   N/A
    //
    // End ----------------------------------------------------------

{
 TRACER("TEXT_OBJECT::append(1 param-char*)");
 time_t mytloc=0;
 time_t theTime;

 theTime = time(mytloc);

 My_String Temp_Text(My_String("\n") + My_String(ctime(&theTime))
     + My_String("\n") + My_String(instring) + My_String("\n"));

 the_text = (char *)(My_String(the_text) + Temp_Text);

 putObject();
};

void TEXT_OBJECT::append(ifstream& input_file)

    // Summary ------------------------------------------------------
    //
    //   append a file as a text object.  stores the file as a
    //   text_object in the database
    //
    // Parameter
    //
    //   input_file
    //
    //   file handle of input file
    //
    // Return Value
    //
    //   N/A
    //
    // End ----------------------------------------------------------
{
 TRACER("TEXT_OBJECT::append(1 param input_file)");
 time_t mytloc=0;
 time_t theTime = 0;

 ostrstream buf;
 char ch;
```

```
 while (buf && input_file.get(ch))
  {
   buf.put(ch);
  }
 buf.put('\0');

 theTime = time(mytloc);
 My_String Temp_Text(My_String("\n") + My_String(ctime(&theTime))
    + My_String("\n") + My_String(buf.str()) + My_String("\n"));
 the_text = (char *)(My_String(the_text) + Temp_Text);

 putObject();
}

void TEXT_OBJECT::text(ostream& outstream)

   // Summary ----------------------------------------------------
   //
   //   output the text_object as a file. dumps the text_object to
   //   the PROTOTYPE environment subdirectory
   //
   // Parameter
   //
   //   outstream
   //
   //   file handle of output file
   //
   // Return Value
   //
   //   N/A
   //
   // End ----------------------------------------------------------

{
 outstream << the_text;
};


Boolean TEXT_OBJECT::rebuildTextFile(char *fileMode)

   // Summary ----------------------------------------------------
   //
   //   output the text_object as a file in "r" - read only or "w" read
   //   and write mode (refer to Unix system manual). dumps the
```

```
    //   text_object to the PROTOTYPE environment subdirectory
    //
    // Parameter
    //
    //   fileMode
    //
    //   "r" - read only. "w" read/write.
    //
    // Return Value
    //
    //   Boolean SUCCESS or FAILURE  -- refers to success of
    //   rebuilding file on the disk.
    //
    // End ----------------------------------------------------
{
 TRACER("TEXT_OBJECT::rebuildTextFile");
 ofstream oFile;
 My_String My_Path = My_String(dirNamePtr) + My_String("/")
     + My_String(the_file_name);
 char *mypath = (char*)My_Path;

 TRACE(mypath);
 if (strcmp(fileMode, "w") == 0 || strcmp(fileMode, "W") == 0)
   {
    oFile.open(mypath,ios::noreplace);

    if (!oFile)
      {
       cerr << "FILE " << the_file_name << " already exists.\n"
            << "Do you want to overwrite the file? Y or N : ";
       char answer;
       cin >> answer;
       if (answer == 'Y' || answer == 'y')
            {
             oFile.open(mypath,ios::out);
             oFile << the_text ;
             oFile.close();
             return SUCCESS;
            }
       else
            {
             return FAILED;
            }
      }
```

```
    else
     {
      oFile << the_text ;
      oFile.close();
     }
   }
 else
   {
    if (the_text)
     cout << the_text;
   }


 return SUCCESS;

}

void TEXT_OBJECT::displayFileName()
   // Summary ---------------------------------------------------
   //
   //   Displays the text_object filename to stdout.
   //
   // Parameter
   //
   //   N/A
   //
   // Return Value
   //
   //   N/A
   //
   // End -------------------------------------------------------
{
 cout << the_file_name << "\n";
}

char *TEXT_OBJECT::getFileName()
   // Summary ---------------------------------------------------
   //
   //   Returns the attribute containing the name of the file
   //   as it was stored on the disk.
   //
   // Parameter
   //
   //   N/A
```

```
    //
    // Return Value
    //
    //   character string containing the file name of the object
    //
    // End ----------------------------------------------------
{
 return the_file_name;
}


char *TEXT_OBJECT::text()

    // Summary ------------------------------------------------
    //
    //   return the contents of the_text
    //
    // Parameter
    //
    //   N/A
    //
    // Return Value
    //
    //   character string pointer with the text in the text_object.
    //
    // End ----------------------------------------------------
{
 return the_text;
};

void TEXT_OBJECT::resetTheText()
    // Summary ------------------------------------------------
    //
    //   Reinitialize the_text attribute to a blank character.
    //
    // Parameter
    //
    //   N/A
    //
    // Return Value
    //
    //   N/A
    //
    // End ----------------------------------------------------
```

strcpy(the_text, "");

```
// File Header ------------------------------------------------------------
  //.............:
  //.Filename.....: thread.h
// Date        : 9/16/91
// Author      : Garry Lewis
//             : Drew Dwyer
// Modified by  : Michael D. O'Loughlin
// Date        : 6/18/92
// Modifications : Extensive modifications have been conducted on almost all of the
/*              original code written by Dwyer and Lewis.  Every function that
                accesses an attribute of an object or the entire object has had
                to be modified. The original code is not even recognizable in
                some functions.
                Because the functionality of the design database system changed
                completely from what Dwyer and Lewis developed each individual
                modification has not been documented. Because of the massive
                changes required I discarded a lot of obsolete code and build a
                new system on the base Dwyer and Lewis had established.
*/
// Compiler     : Glockenspiel C++ 2.1
//
// End header comments ----------------------------------------------------



#ifndef __THREAD_H
#define __THREAD_H

  // SCCS ID follows: will compile to place date/time stamp in
  // object file

  static char thread_h_SccsId[] = "@(#)thread.h 1.3\t9/16/91";

// Contents ---------------------------------------------------
  //
  // THREAD
  //
  // Description
  //
  // Defines class THREAD.
  //
  // End --------------------------------------------------------

  // Interface Dependencies---------------------------------------
```

```
#include <Object.h>
#include <Dictionary.h>
#include <Reference.h>
#include "ReferenceMacros.h"
#include <stream.hxx>

 class V_OBJECT;

// End Interface Dependencies ---------------------------------

 TypeCheckReference(VOListReference, Reference, Dictionary);

class THREAD : public Object
{
 private:
 int current_version; // most recent rev.
 int NumberOfVersions;
 int fromVariation;
 int fromVersion;
 VOListReference the_list;

 public:
 THREAD(APL *theAPL);
 THREAD(char *id);
 virtual void Destroy(Boolean aborted=FALSE);
 virtual Type *getDirectType();
 int getCurrentVersionNum();
 V_OBJECT *current();
 V_OBJECT *version(int version_id);
 void add_object(V_OBJECT *new_object);
 void displayThreadVersions();
 void displayThreadContents();

 void updateNumberOfVersions(int);
 int getNumberOfVersions();
 int previousVersion();
 int previousVariation();
 void setPreviousVersion(int);
 void setPreviousVariation(int);
};

 char *buildThreadName(char*, int);
```

```
// Description ------------------------------------------------
//
// Defines a THREAD class. The class COMPONENT is a derived class
// of Object (i.e. It is a persistent class). A thread may
// contain multiple versions of an COMPONENT, composite or
// configurations.
//
// Constructor
//
// Thread -- APL
//
// ONTOS required constructor.
//
// Thread
//
// Constructs a thread with the given name.
//
// Public Members
//
// Destroy
//
// Used in lieu of a class destructor.
//
// getDirectType
//
// ONTOS required method to return the class type.
//
// getCurrentVersionNum
//
// Returns the version number of the vobject last add to the thread.
//
// current
//
// Returns a pointer to the current vobject in the thread.
//
// version
//
// Returns a pointer to a user designated version of a vobject.
//
// add_Object
//
// inserts a vobject into the thread.
//
```

```
// displayThreadVersions
//
// List the version numbers of vobjects contained in the thread.
//
// displayThreadContents
//
// Displays the version number and description of each vobject in the thread.
//
// End -----------------------------------------------------------------

#endif // __THREAD_H
```

```
// File Header ------------------------------------------------------------
  //.............:
  //.Filename.....: thread.cxx
// Date         : 9/16/91
// Author       : Garry Lewis
//              : Drew Dwyer
// Modified by  : Michael D. O'Loughlin
// Date         : 6/18/92
// Modifications : Extensive modifications have been conducted on almost all of the
/*              original code written by Dwyer and Lewis.  Every function that
                accesses an attribute of an object or the entire object has had
                to be modified. The original code is not even recognizable in
                some functions.
                Because the functionality of the design database system changed
                completely from what Dwyer and Lewis developed each individual
                modification has not been documented. Because of the massive
                changes required I discarded a lot of obsolete code and build a
                new system on the base Dwyer and Lewis had established.
*/
// Compiler     : Glockenspiel C++ 2.1
//
// End header comments ----------------------------------------------------



// SCCS ID follows: will compile to place date/time stamp in object file

static char thread_cxx_SccsId[] = "@(#)thread.cxx       1.3\t9/16/91";

// Contents ---------------------------------------------------------------
  //
  //   THREAD::THREAD  ONTOS constructor
  //   THREAD::THREAD  new instance constructor
  //   THREAD::Destroy
  //   THREAD::getDirectType
  //   THREAD::getCurrentVersionNum
  //   THREAD::current
  //   THREAD::version
  //   THREAD::add_object
  //   THREAD::displayThreadVersions
  //   THREAD::displayThreadContents
  //
  // Description
  //
```

```
// Implementation of class THREAD member functions.
//
// End ------------------------------------------------------------

// TURN ON TRACE-DEBUG
#define DEBUG
#include "debug.h"

#include <strstream.hxx>

#include <GlobalEntities.h>
#include <stream.hxx>

#ifndef __THREAD_H
#include "thread.h"
#endif

#ifndef __VERSIONED_OBJECT_H
#include "versioned_object.h"
#endif

extern Type *THREAD_OType;
extern Type *V_OBJECT_OType;

THREAD::THREAD(APL *theAPL): (theAPL)
{
};

THREAD::THREAD(char *id): (id)

   // Summary ---------------------------------------------------
   //
   // Constructs a persistent THREAD object. A thread contains
   // a list of V_OBJECTS (objects which version), and maintains
   // the most current version from that list of versioned
   // objects.
   //
   // A thread is stored in the ONTOS database and is given
   // visibility. Therefore, only one operator may generate any
   // given thread.
   //
   // It is expected that Variations will inherit from threads
   // with two distinctive bits of information:
   //
```

```
    //   the thread from which it spawned --
    //   the version number from which it originated
    //
    // Parameter
    //
    //   id
    //
    //   passed to the ONTOS database and gives persistence and
    //   ONTOS visibility to that object
    //
    // Return Value
    //
    //   a persistent THREAD in the ONTOS database
    //
    // End ------------------------------------------------------------

{
 TRACER("THREAD::THREAD - char* ");
 initDirectType(THREAD_OType);
 current_version  = 0;
 NumberOfVersions = 0;
 fromVersion      = 0;
 fromVariation    = 0;
 Dictionary *new_list=new Dictionary(OC_integer,
                          V_OBJECT_OType,
                          TRUE,FALSE);
 new_list ->putObject();
 the_list.Reset(new_list, this);
 putObject();
};

void THREAD::Destroy(Boolean aborted)
{
 TRACER("THREAD::Destroy");
 Destroy(aborted);
};

Type *THREAD::getDirectType()

    // Summary --------------------------------------------------------
    //
    //   returns the ONTOS Type for the prototype class.
    //
    // Return value
```

```
    //
    //   A pointer to an ONTOS Type.
    //
    // End ---------------------------------------------------

{
 TRACER("THREAD::getDirectType");
 return THREAD_OType;
};

int THREAD::getCurrentVersionNum()

    // Summary -----------------------------------------------
    //
    //   returns the current version number in the thread of
    //   versioned objects
    //
    // Parameter
    //
    //   N/A
    //
    // Return value
    //
    //   An integer value representing the current version of the
    //   operator/type (as defined by CAPS)
    //
    // End ---------------------------------------------------

{
 TRACER("THREAD::getCurrentVersionNum");
 return current_version;
}


int THREAD::previousVariation()
{
 TRACER("THREAD::previousVariation");
 return fromVariation;
}

int THREAD::previousVersion()
{
 TRACER("THREAD::previousVersion");
 return fromVersion;
```

```
}

void THREAD::setPreviousVersion(int ParentVer)
{
 TRACER("THREAD::setPreviousVersion");
 fromVersion = ParentVer;
}

void THREAD::setPreviousVariation(int ParentVar)
{
 TRACER("THREAD::setPreviousVariation");
 fromVariation = ParentVar;
}

// ************************************************************************

void THREAD::updateNumberOfVersions(int VerNumber)

   // Summary ----------------------------------------------------
   //
   // End --------------------------------------------------------

{
 TRACER("THREAD::updateNumberOfVersions");
   NumberOfVersions = VerNumber;
}


int THREAD::getNumberOfVersions()

   // Summary ----------------------------------------------------
   //
   //
   // End --------------------------------------------------------

{
 TRACER("THREAD::getNumberOfVersions");
 return NumberOfVersions;
}

char* buildThreadName(char *threadName, int variation)
   {
TRACER("buildThreadName");
      if (variation == 0)
```

```
        {
            return threadName;
        }
      else
        {
            int buffsize = strlen(threadName) + 5;
            char *p = new char[buffsize];
            ostrstream ost(p,buffsize);
            ost << threadName << "_" << variation;
            ost.put('\0');
            return p;
        }
   }

//*************************************************************************

V_OBJECT *THREAD::current()

    // Summary ----------------------------------------------------
    //
    //   returns the current versioned object in the thread.
    //
    // Parameter
    //
    //   N/A
    //
    // Return value
    //
    //   A V_OBJECT pointer
    //
    // End ----------------------------------------------------------

{
 TRACER("THREAD::current");
 Dictionary *temp_list= (Dictionary*)the_list.Binding(this);

 V_OBJECT *mytempvo = (V_OBJECT*)(Entity*)(*temp_list)[current_version];
 if (mytempvo)
      {
       TRACE("mytempvo not NULL");
      }
 return mytempvo;

};
```

```
V_OBJECT *THREAD::version(int the_version)

   // Summary ----------------------------------------------------
   //
   //   returns the desired version in the thread of versioned
   //   objects
   //
   // Parameter
   //
   //   N/A
   //
   // Return value
   //
   //   A V_OBJECT pointer
   //
   // End --------------------------------------------------------

{
 TRACER("THREAD::version");
 Dictionary *temp_list= (Dictionary*)the_list.Binding(this);

 V_OBJECT *mytempvo = (V_OBJECT*)(Entity*)(*temp_list)[the_version];

 if (mytempvo) TRACE("mytempvo not NULL");

 return mytempvo;
};


void THREAD::add_object(V_OBJECT *new_object)

   // Summary ----------------------------------------------------
   //
   //   adds a versioned_object to the thread, and updates the
   //   current_version attribute to reflect the newer version
   //
   // Parameter
   //
   //    new_vobject
   //
   //    V_OBJECT pointer
   //
```

```
    // Return Value
    //
    //   N/A
    //
    // End ----------------------------------------------------

{
 TRACER("THREAD::add_object");
 if(!this)
   {
    cerr << "<ERROR: cannot attach a v_object to a null THREAD>\n";
    return;
   }
 if(!new_object)
   {
    cerr << "<ERROR: cannot insert a null v_object into a thread>\n";
    return;
   }
 else
   {
    current_version = current_version + 1;
    Dictionary *temp_list = (Dictionary*)the_list.Binding(this);
    temp_list -> Insert(current_version,(Entity *)new_object);
    temp_list -> putObject();
    putObject();
   }
};

void THREAD::displayThreadVersions()

    // Summary ----------------------------------------------------
    //
    //   Display the versions within a thread to stdout
    //
    // Parameter
    //
    //   N/A
    //
    // Return Value
    //
    //   N/A
    //
    // End ----------------------------------------------------
```

```
{
 TRACER("THREAD::displayThreadVersions");
 Dictionary *temp_list= (Dictionary*)the_list.Binding(this);

 DictionaryIterator next(temp_list);

 while(next.moreData())
  {
   V_OBJECT *temp=(V_OBJECT *)(Entity *)next();
   temp->displayVariationNumber();
   temp->displayVersionNumber();
  };
};


void THREAD::displayThreadContents()
{

 // Summary ----------------------------------------------------
 //
 //   Displays the version and description of each versioned
 //   object of a thread.
 //
 //   NOT USED in current implementation of Design Database.
 //
 // Parameter
 //
 //   N/A
 //
 // Return Value
 //
 //   N/A
 //
 // End ----------------------------------------------------------
 TRACER("THREAD::displayThreadContents");
 Dictionary *temp_list= (Dictionary*)the_list.Binding(this);

 DictionaryIterator next(temp_list);

 while(next.moreData())
  {
   V_OBJECT *temp=(V_OBJECT *)(Entity *)next();
   temp->displayVersionNumber();
   cout << "\n";
```

```
 temp->getDescription();
 cout << "\n";
};
```

```
// File Header ------------------------------------------------------------
 //.............:
 //.Filename.....: tree.h
// Date         : 9/16/91
// Author       : Garry Lewis
//              : Drew Dwyer
// Modified by  : Michael D. O'Loughlin
// Date         : 6/18/92
// Modifications : Extensive modifications have been conducted on almost all of the
/*              original code written by Dwyer and Lewis.  Every function that
                accesses an attribute of an object or the entire object has had
                to be modified. The original code is not even recognizable in
                some functions.
                Because the functionality of the design database system changed
                completely from what Dwyer and Lewis developed each individual
                modification has not been documented. Because of the massive
                changes required I discarded a lot of obsolete code and build a
                new system on the base Dwyer and Lewis had established.
*/
// Compiler     : Glockenspiel C++ 2.1
//
// End header comments ----------------------------------------------------



#ifndef __TREE_H
#define __TREE_H

 // SCCS ID follows: will compile to place date/time stamp in
 // object file

 static char tree_h_SccsId[] = "@(#)tree.h     1.3\t9/16/91";

// Contents ---------------------------------------------------
 //
 //   TREE HEADER
 //
 // Description
 //
 //   Defines class TREE.
 //
 // End --------------------------------------------------------

 // Interface Dependencies -------------------------------------
```

```
#ifndef __TREENODE_H
#include "treenode.h"
#endif

 // End Interface Dependencies ---------------------------------

 class TREE
{
 private:
 char *tree_name;
 TREENODE * theTreeRootNode;

 public:
 TREE(TREENODE *,char *);  // input list and resulting rootnode
  void build_tree(TREENODE *,TREENODE_linkedlist);
 TREENODE *find_treenode(TREENODE_linkedlist, char *);
};

// Description ----------------------------------------------
 //
 //   Defines the TREE class.
 //
 // Constructor
 //
 //   Constructs a multiway tree from a linked list of nodes
 //   identified as operators from reading the subdirectory in
 //   TREENODE class.  In this tree is one unique TREENODE --
 //   the "root".  Once the root is identified, reference to
 //   the tree can be passed to other classes who can then deal
 //   individually with nodes in that tree through the TREENODE
 //   class.
 //
 //   TREE
 //
 //   constructs the tree given a TREENODE object and a character
 //   string pointer to the root operator.
 //
 // Public Members
 //
 //   build_tree
 //
 //   takes in the root TREENODE and a linked list of other
 //   generic operator nodes and builds the multiway tree using
```

```
//    pointers and lists of children nodes.
//
//    find_treenode
//
//    given a character string of a TREENODE and a linked list of
//    TREENODES, search the linked list and return a TREENODE
//    pointer on a match.  Return a NULL pointer if fails.
//
// End -------------------------------------------------------

#endif  // __TREE_H
```

```
// File Header ------------------------------------------------------------
 //.............:
 //.Filename.....: tree.cxx
// Date        : 9/16/91
// Author      : Garry Lewis
//             : Drew Dwyer
// Modified by  : Michael D. O'Loughlin
// Date        : 6/18/92
// Modifications : Extensive modifications have been conducted on almost all of the
/*              original code written by Dwyer and Lewis.  Every function that
                accesses an attribute of an object or the entire object has had
                to be modified. The original code is not even recognizable in
                some functions.
                Because the functionality of the design database system changed
                completely from what Dwyer and Lewis developed each individual
                modification has not been documented. Because of the massive
                changes required I discarded a lot of obsolete code and build a
                new system on the base Dwyer and Lewis had established.
*/
// Compiler     : Glockenspiel C++ 2.1
//
// End header comments ----------------------------------------------------



// SCCS ID follows: will compile to place date/time stamp in
// object file

static char tree_cxx_SccsId[] = "@(#)tree.cxx   1.3\t9/16/91";

// Contents ---------------------------------------------------
//
//   TREE::TREE
//   TREE::build_tree
//   TREE::find_treenode
//
// Description
//
//   IMPLEMENTS class TREE CONSTRUCTORS.
//
// End --------------------------------------------------------

 // Interface Dependencies -------------------------------------
```

```
// TURN ON TRACE-DEBUG
#define DEBUG
#include "debug.h"


#include <Database.h>
#include <stream.hxx>

extern "C--"
{
#include <stddef.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <dirent.h>
#include <sys/stat.h>
#include <time.h>
}

#ifndef __TREE_H
#include "tree.h"
#endif

#ifndef __QUEUE_H
#include "queue.h"
#endif

#ifndef __NODESUPPORT_H
#include "nodesupport.h"
#endif

// ----------  End Interface Dependencies  ------------------

TREE::TREE(TREENODE *future_root,char *treename)

    // Summary ---------------------------------------------------
    //
    //   Constructor
    //
    // Parameter
    //
    //   future_root
    //
    //   TREENODE pointer containing the future root of the multiway
```

```
   //   tree
   //
   //   treename
   //
   //   character string -- the same name as the root operator
   //
   // Return Value
   //
   //   A construted multiway tree reflecting the nodes which
   //   exist in the subdirectory and which will be checked into
   //   the design database
   //
   // End ---------------------------------------------------------

{
 tree_name = new char [strlen(treename)+1];
 strcpy(tree_name,treename);
 theTreeRootNode = future_root;
}

TREENODE *TREE::find_treenode(TREENODE_linkedlist list_to_search,
                    char *node_name)

   // Summary -----------------------------------------------------
   //
   //   find_treenode
   //
   // Parameter
   //
   //   list_to_search
   //
   //   a linked list of TREENODES
   //
   //   node_name
   //
   //   the operator/type to search for (i.e. - the name of the
   //   operators filename MINUS the .ps, .graph, .imp.psdl,
   //   .spec.psdl, .a extension
   //
   // Return Value
   //
   //   TREENODE if found -- NULL pointer if not found.
   //
   // End ---------------------------------------------------------
```

```
{
 slist_iterator list_iterator(list_to_search);
 TREENODE *tnode;
 while (tnode=list_iterator())
  if (strcmp(tnode->getname(),node_name)==0)
   return tnode;
 return NULL;

}

void TREE::build_tree(TREENODE *root_node,TREENODE_linkedlist search_list)

   // Summary ---------------------------------------------------
   //
   //   Builds a multiway tree containing the nodes in the directory
   //   and information required to determine whether a new version
   //   of the node must be created in the ONTOS Design Database.
   //
   // Parameter
   //
   //   root_node
   //
   //   the unique TREENODE which is the root of this multiway tree
   //
   //   search_list
   //
   //   a list of operators in the subdirectory pointed to by the
   //   environment variable PROTOTYPE
   //
   // Return Value
   //
   //   N/A
   //
   // End -------------------------------------------------------

{
 TREENODE * nodeptr;
 TREENODE * temp_TREENODE_ptr;
 TREENODE * new_TREENODE_ptr;
 // create the queue inorder to construct the tree
 TREENODE_queue tree_node_queue;
 tree_node_queue.put(root_node);
```

```
// now the queue has the first TREENODE on it
while (!tree_node_queue.empty())
  {
   temp_TREENODE_ptr  = tree_node_queue.get();
   // now iterate through search_list , look for NODES whose associated
   // strings are "proper" superstrings of temp_TREENODE_ptr->operator_name
   //  If they are create a TREENODE for these child nodes,
   //  put the TREENODE in queue as well as in temp_TREENODE_ptr->children
   //  list.
   slist_iterator OperatorPtr(search_list);
   while (nodeptr=OperatorPtr())
    {
     if (proper_super_NODE_check(nodeptr,temp_TREENODE_ptr->getname()))
      //  create the new TREENODE
      {
       new_TREENODE_ptr = new TREENODE(nodeptr, temp_TREENODE_ptr);
       tree_node_queue.put(new_TREENODE_ptr);
       temp_TREENODE_ptr->insertChildNode(new_TREENODE_ptr);
      }
    }
  }
}
```

```
// File Header ---------------------------------------------------------
 //.............:
 //.Filename.....: treenode.h
// Date         : 9/16/91
// Author       : Garry Lewis
//              : Drew Dwyer
// Modified by  : Michael D. O'Loughlin
// Date         : 6/18/92
// Modifications : Extensive modifications have been conducted on almost all of the
/*              original code written by Dwyer and Lewis.  Every function that
                accesses an attribute of an object or the entire object has had
                to be modified. The original code is not even recognizable in
                some functions.
                Because the functionality of the design database system changed
                completely from what Dwyer and Lewis developed each individual
                modification has not been documented. Because of the massive
                changes required I discarded a lot of obsolete code and build a
                new system on the base Dwyer and Lewis had established.
*/
// Compiler     : Glockenspiel C++ 2.1
//
// End header comments -----------------------------------------------



#ifndef __TREENODE_H
#define __TREENODE_H

 // SCCS ID follows: will compile to place date/time stamp in
 // object file

 static char treenode_h_SccsId[] = "@(#)treenode.h     1.3\t9/16/91";

// Contents ----------------------------------------------------
 //
 //   TREENODE HEADER
 //
 // Description
 //
 //   Defines class TREENODE.
 //
 // End ---------------------------------------------------------

 // Interface Dependencies ---------------------------------
```

```
#ifndef __QUEUE_H
#include "queue.h"
#endif

#ifndef __VERSIONED_OBJECT_H
#include "versioned_object.h"
#endif

 // End Interface Dependencies ------------------------------
 class TREENODE;

class TREENODE
{
 private:
 char *tree_node_name;
 char *node_name;
 long  timestamp;
 int level;
 TREENODE_linkedlist ChildrenList;
 TREENODE * ParentNode;

 public:
 TREENODE(char *,TREENODE *);
 TREENODE(TREENODE *,TREENODE *);
 void updatetimestamp(long time);
 char *getname();
 void insertChildNode(TREENODE *);
 TREENODE_linkedlist getChildren();
 TREENODE *getParentNode();
 char *get_asc_time();
 int getlevel();
 long get_long_time();
 void list_subtree();
 void checkin_subtree(V_OBJECT *);
 V_OBJECT *checkin_node(V_OBJECT *);
};

// Description ---------------------------------------------
 //
 //   TREENODE
 //
 //   TREENODE
 //
```

```
//  Constructor - Builds a treenode to be a node plus pointer
//              information for building a multiway tree.
//
//  updatetimestamp
//
//  used to compare the most current filestamp of the group of
//  five files in a versioned object from the disk directory
//  pointed to by the CAPS environment variable PROTOTYPE to
//  the locktime of the matching versioned object stored
//  in the Design database.
//
//  getname
//
//  returns the character string name of the treenode.
//
//  insertChildNode
//
//  used to insert a node as a child of the current treenode
//
//  getChildren
//
//  returns the linked list of children of this node
//
//  getParentNode
//
//  returns the parent of this node
//
//  get_asc_time
//
//  returns the ctime function for the treenode timestamp attribute
//
//  get_level
//
//  returns the integer level (0 = root, 1 is removed from root
//  1 level, etcetera
//
//  get_long_time
//
//  returns the timestamp from the treenode as a long that can
//  be used in a comparison in the checkin_node function
//
//  list_subtree
//
//  used for debugging.  Lists the multiway tree
```

```
//
//   checkin_subtree
//
//   after the multiway tree is built, this function launches the
//   recursion which does the bulk of the work.
//
//   checkin_node
//
//   the function which compares the TREENODE (as read from
//   disk) to threads in the database.  If a match is found,
//   locktimes are compared to timestamp and if timestamp is
//   more recent, then a new versioned object is created for
//   the database.  All version links are set up in this
//   function
//
// End Description ---------------------------------------------


#endif  // header file



    // File Header ------------------------------------------------------------
 //.............:
 //.Filename.....: treenode.cxx
// Date        : 9/16/91
// Author      : Garry Lewis
//             : Drew Dwyer
// Modified by  : Michael D. O'Loughlin
// Date        : 6/18/92
// Modifications : Extensive modifications have been conducted on almost all of the
/*              original code written by Dwyer and Lewis.  Every function that
                accesses an attribute of an object or the entire object has had
                to be modified. The original code is not even recognizable in
                some functions.
                Because the functionality of the design database system changed
                completely from what Dwyer and Lewis developed each individual
                modification has not been documented. Because of the massive
                changes required I discarded a lot of obsolete code and build a
                new system on the base Dwyer and Lewis had established.
*/
// Compiler     : Glockenspiel C++ 2.1
//
// End header comments ----------------------------------------------------
```

```
// SCCS ID follows: will compile to place date/time stamp in
// object file

static char treenode_cxx_SccsId[] = "@(#)treenode.cxx   1.3\t9/16/91";
// Contents ----------------------------------------------------
//
//   TREENODE::TREENODE
//   TREENODE::TREENODE
//   TREENODE::updatetimestamp
//   TREENODE::getname
//   TREENODE::insertChildNode
//   TREENODE::getChildren
//   TREENODE::getparentNode
//   TREENODE::get_asc_time
//   TREENODE::getlevel
//   TREENODE::get_long_time
//   TREENODE::list_subtree
//   TREENODE::checkin_subtree
//   TREENODE::checkin_node
//
// Description
//
//   IMPLEMENTS class TREENODE CONSTRUCTORS and methods.
//
// End ---------------------------------------------------------

// Interface Dependencies --------------------------------------

// TURN ON TRACE-DEBUG
#define DEBUG
#include "debug.h"

#include "My_String.h
#include <Directory.h>
#include <stream.hxx>
#include <strstream.hxx>

extern "C--"
{
#include <stddef.h>
#include <stdlib.h>
```

```
#include <stdio.h>
#include <string.h>
#include <dirent.h>
#include <sys/stat.h>
#include <time.h>
#include <unistd.h>
}
#ifndef __THREAD_H
#include "thread.h"
#endif

#ifndef __COMPONENT_H
#include "component.h"
#endif

#ifndef __TEXT_OBJECT_H
#include "text_object.h"
#endif

#ifndef __TREENODE_H
#include "treenode.h"
#endif

#ifndef __DDBDEFINES_H
#include "ddbdefines.h"
#endif

#ifndef __VOBJECTFUNC_H
#include "vobjectfunc.h"
#endif
// ---------- End Interface Dependencies ----------

extern char *dirNamePtr;


TREENODE::TREENODE(char *name, TREENODE* future_parent)
{
  tree_node_name = new char[strlen(name)+1];
  strcpy(tree_node_name,name);
  level=0;
  timestamp = 0;
  ParentNode =future_parent;
}
```

```
TREENODE::TREENODE(TREENODE *inc_data, TREENODE* future_parent)
{
 tree_node_name = new char[strlen(inc_data->getname())+1];
 strcpy(tree_node_name,inc_data->getname());
 if ((future_parent)!=NULL)
  level=future_parent->getlevel()+1;
 else
  level=1;
 timestamp = inc_data->get_long_time();
 ParentNode = future_parent;
}

void TREENODE::updatetimestamp(long time)
{
 timestamp=time;
}

char *TREENODE::getname()
{
 return tree_node_name;
}

void TREENODE::insertChildNode(TREENODE *new_child)
{
 ChildrenList.insert(new_child);
}

TREENODE_linkedlist TREENODE::getChildren()
{
 return ChildrenList;
}

TREENODE *TREENODE::getParentNode()
{
 return ParentNode;
}

char *TREENODE::get_asc_time()
{
 return ctime(&timestamp);
}

int TREENODE::getlevel()
{
```

```
 return level;
}

long TREENODE::get_long_time()
{
 return timestamp;
}

void TREENODE::list_subtree()
{
 TRACER("TREENODE::list_subtree");
 slist_iterator ChildrenPtr(ChildrenList);
 TREENODE *tnode;
 for (;;)        // recursive call inside infinite for loop
  {
   tnode = ChildrenPtr();
   if (tnode!=NULL)
    {
     char *name=tnode->getname();
     int level = tnode->getlevel();
     char *asctime = tnode->get_asc_time();
     cout << level << "-->" << name << "time: "<<asctime;
     tnode->list_subtree();              //preorder
    }
   else
    break;      // breaks when end of list reached.
  }
}



void TREENODE::checkin_subtree(V_OBJECT *new_parent)
{
 TRACER("TREENODE::checkin_subtree");
 slist_iterator ChildrenPtr(ChildrenList);
 TREENODE *tnode;
 V_OBJECT *parent;
 THREAD *NewThreadPtr  =  (THREAD *)0;
 THREAD *threadPtr     =  (THREAD *)0;
 V_OBJECT *new_object  =  (V_OBJECT *)0;
 V_OBJECT *var_temp    =  (V_OBJECT *)0;
 V_OBJECT *original_parent = (V_OBJECT *)0;
 V_OBJECT *child_exists = (V_OBJECT *)0;
 List *children        =  (List *)0;
```

```
char *threadName     =  (char *)0;

for (;;)        // recursive call inside infinite for loop
 {
  tnode = ChildrenPtr();

  if (tnode!=NULL)
   {
    parent = tnode->checkin_node(new_parent);
    tnode->checkin_subtree(parent);    //preorder
   }
  else
   {
   break;      // breaks when end of list reached.
   }
 }

if (new_parent->Needs_Updating() && !new_parent->just_created())
    {
     int variationNum;
     int versionNum = 1;
     threadPtr = new_parent->getThread();
     if (new_parent -> getVersionNumber() <
          threadPtr -> getCurrentVersionNum())
       {
        threadName = buildThreadName(tree_node_name,1);
        THREAD *tempThread = ((THREAD *)OC_lookup(threadName));
        var_temp = tempThread -> version(1);
        variationNum = var_temp -> getNumberOfVariations() + 1;
        char *newThread = buildThreadName(tree_node_name,variationNum);
        NewThreadPtr = new THREAD(newThread);
        NewThreadPtr->setPreviousVariation(new_parent->getVariationNumber());
        NewThreadPtr->setPreviousVersion(new_parent->getVersionNumber());
        NewThreadPtr->updateNumberOfVersions(versionNum);
        NewThreadPtr->putObject();
        new_parent->Reset_Update();
        original_parent = new_parent->getParent();
        var_temp-> incrementNumberOfVariations();
        new_object = new V_OBJECT(versionNum);
        new_object-> copyChildren(new_parent);
        new_object-> updateVariationNumber(variationNum);
        new_object-> connect_vobject_to_thread(NewThreadPtr);
        new_object-> addVariationThread(NewThreadPtr, variationNum);
        var_temp-> addVariation(new_object, variationNum);
```

```
    new_object->setNodeName(new_parent->getNodeName());
    new_object->setParent(original_parent);
    new_object-> addCOMPONENTNode(new_parent->getCOMPONENT());
    new_object->reset_created();
    new_object->Reset_Update();
    if (original_parent)
     {
      child_exists =
       original_parent->check_for_child(new_object->getNodeName());
      if (child_exists)
       {
        original_parent->deleteChildNode(child_exists);
        original_parent->addChildNode(new_object);
       }
     }
    children = new_object->getChildren();
    if (children)
     {
      children->putObject();
     }
    var_temp->putObject();
    new_object->putObject();
    NewThreadPtr->add_object(new_object);
  }
 else
  {
    var_temp = threadPtr -> version(1);
    versionNum = new_parent->getVersionNumber() + 1;
    variationNum = new_parent->getVariationNumber();
    new_object = new V_OBJECT(versionNum);
    new_object-> updateVariationNumber(variationNum);
    new_object-> connect_vobject_to_thread(threadPtr);
    new_object-> copyChildren(new_parent);
    new_object-> setNodeName(new_parent->getNodeName());
    original_parent = new_parent->getParent();
    new_object-> setParent(original_parent);
    new_object-> addVariationThread(threadPtr, variationNum);
    new_object-> addCOMPONENTNode(new_parent->getCOMPONENT());
    new_object->reset_created();
    new_object->Reset_Update();
    child_exists =
      original_parent->check_for_child(new_object->getNodeName());
    if (child_exists)
     {
```

```
            original_parent->deleteChildNode(child_exists);
            original_parent->addChildNode(new_object);
           }
          children = new_object->getChildren();
          if (children)
           {
            children->putObject();
           }
          var_temp->putObject();
          threadPtr->add_object(new_object);
          new_object->putObject();
         }
      }
     else
      {
      if (new_parent->just_created())
        {
         original_parent = new_parent->getParent();

         if (original_parent)
          {
         child_exists =
           original_parent->check_for_child(new_parent->getNodeName());
           if (child_exists)
            {
             original_parent->deleteChildNode(child_exists);
            }
           original_parent->addChildNode(new_parent);
          }
         new_parent->reset_created();
         new_parent->Reset_Update();
         children = new_parent->getChildren();
         if (children)
          {
           children->putObject();
          }
         new_parent->putObject();
        }
      }
}


V_OBJECT *TREENODE::checkin_node(V_OBJECT *future_parent)
{
```

```
TRACER("TREENODE::checkin_node");
ifstream psfile;
ifstream graphfile;
ifstream impfile;
ifstream specfile;
ifstream sourcefile;

Boolean create_new_vobject = FALSE;
Boolean new_thread_created = FALSE;
Boolean lockTime_timeStamp = FALSE;
Boolean new_spec_object    = FALSE;
Boolean new_graph_object   = FALSE;
Boolean new_imp_object     = FALSE;
Boolean new_ps_object      = FALSE;
Boolean new_source_object  = FALSE;
Boolean SPEC_FILE          = FALSE;
Boolean GRAPH_FILE         = FALSE;
Boolean IMP_FILE           = FALSE;
Boolean PS_FILE            = FALSE;
Boolean SOURCE_FILE        = FALSE;

TEXT_OBJECT * new_graphfile_object   = new TEXT_OBJECT();
TEXT_OBJECT * new_sourcefile_object  = new TEXT_OBJECT();
TEXT_OBJECT * new_impfile_object     = new TEXT_OBJECT();
TEXT_OBJECT * new_psfile_object      = new TEXT_OBJECT();
TEXT_OBJECT * new_specfile_object    = new TEXT_OBJECT();

char *psfilename;
char *graphfilename;
char *impfilename;
char *specfilename;
char *sourcefilename;

char *COMPONENT_psfilename;
char *COMPONENT_graphfilename;
char *COMPONENT_impfilename;
char *COMPONENT_specfilename;
char *COMPONENT_sourcefilename;

psfilename = (char*)(My_String(dirNamePtr) + My_String("/") +
    My_String(tree_node_name) + My_String(".ps") );

graphfilename = (char*)(My_String(dirNamePtr) + My_String("/") +
    My_String(tree_node_name) + My_String(".graph") );
```

```
specfilename = (char*)(My_String(dirNamePtr) + My_String("/") +
    My_String(tree_node_name) + My_String(".spec.psdl") );

impfilename = (char*)(My_String(dirNamePtr) + My_String("/") +
    My_String(tree_node_name) + My_String(".imp.psdl") );

sourcefilename = (char*)(My_String(dirNamePtr) + My_String("/") +
    My_String(tree_node_name) + My_String(".a") );

COMPONENT_psfilename = (char*)(My_String(tree_node_name) +
    My_String(".ps"));

COMPONENT_graphfilename = (char*)(My_String(tree_node_name) +
    My_String(".graph"));

COMPONENT_impfilename = (char*)(My_String(tree_node_name) +
    My_String(".imp.psdl"));

COMPONENT_specfilename = (char*)(My_String(tree_node_name) +
    My_String(".spec.psdl"));

COMPONENT_sourcefilename = (char*)(My_String(tree_node_name) +
    My_String(".a"));

psfile.open(psfilename);
graphfile.open(graphfilename);
impfile.open(impfilename);
specfile.open(specfilename);
sourcefile.open(sourcefilename);


THREAD *NewThreadPtr  = (THREAD *)0;
THREAD *threadPtr     = (THREAD *)0;
V_OBJECT *vobjectPtr  = (V_OBJECT *)0;
V_OBJECT *new_vobject = (V_OBJECT *)0;
V_OBJECT *var_temp    = (V_OBJECT *)0;
V_OBJECT *SameVarPtr  = (V_OBJECT *)0;
DDBControlData *Working_VarVerPtr;
char *p = (char *)0;
char *pp = (char *)0;
char *oldPsText = (char *)0;
char *oldSourceText = (char *)0;
char *oldSpecText = (char *)0;
```

```
char *oldImpText = (char *)0;
char *oldGraphText = (char *)0;
long vobject_locktime = 0;
int versionNum = 1;
int variationNum = 0;

p = buildThreadName(tree_node_name,1);
if (threadPtr = ((THREAD *)OC_lookup(p)))
 {
  char *ddbcontrolpath = new char [strlen(dirNamePtr) + strlen(tree_node_name) + 13];
  strcpy (ddbcontrolpath,dirNamePtr);
  strcat (ddbcontrolpath,"/ddbCtrlData.");
  strcat (ddbcontrolpath,tree_node_name);
  Working_VarVerPtr = getddbControlFile(ddbcontrolpath);

 if(Working_VarVerPtr)
  {
  var_temp = threadPtr -> version(1);
  V_OBJECT *new_var_temp =
        var_temp -> variation(Working_VarVerPtr->variation);
  threadPtr = new_var_temp -> getThread();
  vobjectPtr = threadPtr -> version(Working_VarVerPtr->version);
  char *removeControlFile = new char[3 + strlen(ddbcontrolpath)];
  unlink (ddbcontrolpath);
  }
 else
  {
  cerr << "Getting default Variation, Version\n";
  vobjectPtr = getDefaultVObject(threadPtr);
  cerr << "Default variation being added to variation "
     <<  vobjectPtr-> getVariationNumber()
     << " version " << vobjectPtr-> getVersionNumber() << "\n";
  }
  vobject_locktime = vobjectPtr ->getLockTime(); // return locktime
 }
else
 {
  NewThreadPtr = new THREAD(p);
  create_new_vobject = TRUE;
  new_thread_created = TRUE;
  vobject_locktime = TRUE;
 }

if (vobjectPtr)
```

```
{
 COMPONENT *SCOMPONENTPtr;
 SCOMPONENTPtr = vobjectPtr->getCOMPONENT();
 oldSpecText = SCOMPONENTPtr-> getTEXTPtr(SPEC_EXT);
 COMPONENT *SRCOMPONENTPtr;
 SRCOMPONENTPtr = vobjectPtr->getCOMPONENT();
 oldSourceText = SRCOMPONENTPtr-> getTEXTPtr(SOURCE_EXT);
 COMPONENT *GCOMPONENTPtr;
 GCOMPONENTPtr = vobjectPtr->getCOMPONENT();
 oldGraphText = GCOMPONENTPtr-> getTEXTPtr(GRAPH_EXT);
 COMPONENT *ICOMPONENTPtr;
 ICOMPONENTPtr = vobjectPtr->getCOMPONENT();
 oldImpText = ICOMPONENTPtr-> getTEXTPtr(IMP_EXT);
 COMPONENT *PCOMPONENTPtr;
 PCOMPONENTPtr = vobjectPtr->getCOMPONENT();
 oldPsText = PCOMPONENTPtr-> getTEXTPtr(PS_EXT);
}

if (specfile)
{
 char *newText;
 SPEC_FILE = TRUE;
  new_specfile_object->append(COMPONENT_specfilename,specfile);
  newText = new_specfile_object->text();
  if (!oldSpecText)
   {
    new_spec_object = TRUE;
   }
  else
   {
    if (strcmp(newText, oldSpecText) == 0)
     {
      new_spec_object = FALSE;
     }
    else
     {
      new_spec_object = TRUE;
     }
   }
 }

if (sourcefile)
{
 char *newText;
```

```
  SOURCE_FILE = TRUE;
  new_sourcefile_object->append(COMPONENT_sourcefilename,sourcefile);
  newText = new_sourcefile_object->text();
  if (!oldSourceText)
   {
    new_source_object = TRUE;
   }
  else
   {
    if (strcmp(newText, oldSourceText) == 0)
     {
      new_source_object = FALSE;
     }
    else
     {
      new_source_object = TRUE;
     }
   }
 }

if (graphfile)
 {
  char *newText;
  GRAPH_FILE = TRUE;
  new_graphfile_object->append(COMPONENT_graphfilename,graphfile);
  newText = new_graphfile_object->text();
  if (!oldGraphText)
   {
    new_graph_object = TRUE;
   }
  else
   {
    if (strcmp(newText, oldGraphText) == 0)
     {
      new_graph_object = FALSE;
     }
    else
     {
     new_graph_object = TRUE;
     }
   }
 }
```

```
if (impfile)
 {
  char *newText;
  IMP_FILE = TRUE;
  new_impfile_object->append(COMPONENT_impfilename,impfile);
  newText = new_impfile_object->text();
  if (!oldImpText)
   {
    new_imp_object = TRUE;
   }
  else
   {
    if (strcmp(newText, oldImpText) == 0)
     {
      new_imp_object = FALSE;
     }
    else
     {
      new_imp_object = TRUE;
     }
   }
 }

if (psfile)
 {
  char *newText;
  PS_FILE = TRUE;
  new_psfile_object->append(COMPONENT_psfilename,psfile);
  newText = new_psfile_object->text();
  if (!oldPsText)
   {
    new_ps_object = TRUE;
   }
  else
   {
    if (strcmp(newText, oldPsText) == 0)
     {
      new_ps_object = FALSE;
     }
    else
     {
      new_ps_object = TRUE;
     }
   }
```

```
}

if ( (new_spec_object || new_graph_object || new_imp_object ||
      new_source_object || new_ps_object)  && vobjectPtr)
 {
  Boolean last_operation_was_checkin = vobjectPtr->get_last_operation();
  if (last_operation_was_checkin)
   {
    cout << "Last operation was VAA ... Preventing duplicates\n";
    return vobjectPtr;
   }
  else
   {
    if (vobjectPtr -> getVersionNumber() <
        threadPtr -> getCurrentVersionNum())
     {
      variationNum = var_temp -> getNumberOfVariations() + 1;
      p = buildThreadName(tree_node_name,variationNum);
      NewThreadPtr = new THREAD(p);
      NewThreadPtr->setPreviousVariation(vobjectPtr->getVariationNumber());
      NewThreadPtr->setPreviousVersion(vobjectPtr->getVersionNumber());
      NewThreadPtr->putObject();
      new_thread_created = TRUE;
      create_new_vobject = TRUE;
     }
    else
     {
      versionNum = vobjectPtr -> getVersionNumber() + 1;
      variationNum = vobjectPtr -> getVariationNumber();
      create_new_vobject = TRUE;
     }
   }
 }


if (vobjectPtr)
 {
  vobjectPtr->reset_created();
  vobjectPtr->releaseLock();
  vobjectPtr->resetLastOpTrue();
  vobjectPtr->resetVisitedFlag();   // Just for good measure, before pass 2
  vobjectPtr->putObject();
 }
```

```
if (create_new_vobject)            // if compare says I need a new
 {
   new_vobject= new V_OBJECT(versionNum);

   if (vobjectPtr)
    {
     new_vobject->copyChildren(vobjectPtr);
    }

   COMPONENT *new_COMPONENT=new COMPONENT();

   if (new_thread_created)
    {
     if (var_temp)
      {
       var_temp-> incrementNumberOfVariations();
       int nextVariation = var_temp-> getNumberOfVariations();
       new_vobject-> updateVariationNumber(nextVariation);
       new_vobject-> connect_vobject_to_thread(NewThreadPtr);
       new_vobject-> addVariationThread(NewThreadPtr, nextVariation);
       var_temp-> addVariation(new_vobject, nextVariation);
       var_temp->putObject();
       NewThreadPtr->displayThreadVersions();
      }
     else
      {
       new_vobject-> incrementNumberOfVariations();
       int nextVariation = new_vobject-> getNumberOfVariations();
       new_vobject-> updateVariationNumber(nextVariation);
       new_vobject-> addVariationThread(NewThreadPtr, nextVariation);
       new_vobject-> addVariation(new_vobject, nextVariation);
       new_vobject->connect_vobject_to_thread(NewThreadPtr);
      }
    }
   else
    {
     new_vobject-> updateVariationNumber(variationNum);
     new_vobject->connect_vobject_to_thread(threadPtr);
    }

   if (PS_FILE)
    {
     new_COMPONENT ->addTextObject(new_psfile_object);
    }
```

```
    if (GRAPH_FILE)
      {
       new_COMPONENT ->addTextObject(new_graphfile_object);
      }

    if (SPEC_FILE)
      {
       new_COMPONENT ->addTextObject(new_specfile_object);
      }

    if (IMP_FILE)
      {
       new_COMPONENT ->addTextObject(new_impfile_object);
      }

    if (SOURCE_FILE)
      {
       new_COMPONENT ->addTextObject(new_sourcefile_object);
      }

    new_vobject->addCOMPONENTNode(new_COMPONENT);

    if (new_thread_created)
      {
       NewThreadPtr->updateNumberOfVersions(versionNum);
       NewThreadPtr->add_object(new_vobject);
      }
    else
      {
       threadPtr->updateNumberOfVersions(versionNum);
       threadPtr->add_object(new_vobject);
      }

  }

psfile.close();
graphfile.close();
impfile.close();
specfile.close();
sourcefile.close();
unlink(psfilename);
unlink(sourcefilename);
unlink(impfilename);
```

```
 unlink(specfilename);
 unlink(graphfilename);
 if (create_new_vobject)
   {
    new_vobject->setNodeName(getname());
    new_vobject->setParent(future_parent);
    new_vobject->Update_Parent();
    return new_vobject;          // return new version of vobject as parent
   }
 else
   {
    vobjectPtr->setParent(future_parent);
    return vobjectPtr;           // return old version of vobject as parent
   }
}
```

```
// File Header ------------------------------------------------------------
 //.............:
 //.Filename.....: versioned_object.h
// Date        : 9/16/91
// Author      : Garry Lewis
//             : Drew Dwyer
// Modified by  : Michael D. O'Loughlin
// Date        : 6/18/92
// Modifications : Extensive modifications have been conducted on almost all of the
/*              original code written by Dwyer and Lewis.  Every function that
                accesses an attribute of an object or the entire object has had
                to be modified. The original code is not even recognizable in
                some functions.
                Because the functionality of the design database system changed
                completely from what Dwyer and Lewis developed each individual
                modification has not been documented. Because of the massive
                changes required I discarded a lot of obsolete code and build a
                new system on the base Dwyer and Lewis had established.
*/
// Compiler     : Glockenspiel C++ 2.1
//
// End header comments ----------------------------------------------------



#ifndef __VERSIONED_OBJECT_H
#define __VERSIONED_OBJECT_H

 // SCCS ID follows: will compile to place date/time stamp in object file

 static char versioned_object_h_SccsId[] = "@(#)versioned_object.h     1.3\t9/16/91";

#include <Object.h>
#include <List.h>
#include <Dictionary.h>
#include <Reference.h>
#include "ReferenceMacros.h"
#include <stream.hxx>

extern "C--"
{
#include <sys/time.h>
#include <sys/types.h>
#include <string.h>
```

```
}

#ifndef __THREAD_H
#include "thread.h"
#endif

#ifndef __COMPONENT_H
#include "component.h"
#endif

#define DEFAULT_VER 1

TypeCheckReference(VariationReference, Reference, Dictionary);
TypeCheckReference(NewVariationThreads, Reference, Dictionary);
TypeCheckReference(DescReference, Reference, TEXT_OBJECT);
TypeCheckReference(COMPONENTObjReference, Reference, COMPONENT);
TypeCheckReference(ChildVObjReference, Reference, List);
TypeCheckReference(ThreadObjReference, Reference, THREAD);

class V_OBJECT : public Object
{
 TypeCheckReference(ParentObjReference, Reference, V_OBJECT);

 private:
 int theVersionNumber;
 int theVariationNumber;
 int NumberOfVariations;
 time_t creationDate;
 time_t lockTime;
 char *node_name;
 char *creator;
 char *worker;
 Boolean visited;                    // for navigation through tree structure
 Boolean last_op_checkin;            // guards against double checkin
 Boolean needsUpdating;
 Boolean created;
 DescReference theDescriptionPtr;
 NewVariationThreads NewVariations;
 VariationReference VariationList;
 ThreadObjReference theThreadPtr;
 COMPONENTObjReference theCOMPONENTPtr;
 ChildVObjReference theChildPtr;
 ParentObjReference theParentPtr;
```

```
public:

V_OBJECT(APL *);
V_OBJECT(int= DEFAULT_VER);
void Destroy(Boolean aborted=FALSE);
Type *getDirectType();
void connect_vobject_to_thread(THREAD *);
void setParent(V_OBJECT *);
void setNodeName(char *);
char *getNodeName();
void getVObjName();
char *getName();
void resetVisitedFlag();
void setVisitedFlag();
Boolean getVisitedFlag();
void getVObjComponentsName();

void displayVariationNumber();
int getVariationNumber();
void updateVariationNumber(int);
V_OBJECT *variation(int);
void displayNumberOfVariations();
int getNumberOfVariations();
void incrementNumberOfVariations();
void addVariation(V_OBJECT *, int);
THREAD *getThread();
void addVariationThread(THREAD *, int);
void copyChildren(V_OBJECT*);
V_OBJECT *check_for_child(char *);
Boolean Needs_Updating();
void set_created();
void reset_created();
Boolean just_created();
void Update_Parent();
void Reset_Update();

void displayVersionNumber();
int getVersionNumber();
void dumpVObjSummary();
time_t setCreationDate();
time_t getCreationDate();
void setLock();
char *getWorker();
char *getCreator();
```

```
  void setWorker();
  void resetLastOpTrue();
  void resetLastOpFalse();
  Boolean get_last_operation();       // returns true if last op was checkin
  int releaseLock();
  time_t getLockTime();
  void getDescription();
  void listChildren();
  void longlistOperatorNames();
  void listOperatorNames();
  void updateDescription(char *, ifstream &);
  void addCOMPONENTNode(COMPONENT *);
  void deleteChildNode(V_OBJECT *);
  void addChildNode(V_OBJECT *);
  V_OBJECT *getParent();
  COMPONENT *getCOMPONENT();
  void dumpSubtree(char *);
  void releaseLockSubtree();
  List *getChildren();
  Boolean getChildPtr();
  Boolean checkoutCOMPONENTNode(char *);
  ~V_OBJECT() { Destroy(FALSE); };
};

// Description ----------------------------------------------
  //   V_OBJECT
  //   V_OBJECT
  //
  //   Constructors - builds a persistent object in the Ontos
  //   database.
  //
  //   Destroy
  //
  //   Required by Ontos.  Every persistent object must have a
  //   destroy function.
  //
  //   getDirectType
  //
  //   Returns an Ontos Type
  //
  //   connect_vobject_to_thread
  //
  //   Connects a vobject to a thread bearing it's name.
  //
```

```
//  setParent
//
//  Used to establish links (Transparent References as Ontos
//  calls them) in the Design Database reflecting the
//  decomposition of CAPS operators/types.
//
//  setNodeName
//
//  NodeName is maintained as a separate character string field.
//
//  getNodeName
//
//  get the shorter NodeName
//
//  getVObjName
//
//  Displays the versioned object's name to stdout
//
//  getName
//
//  returns the character string pointer of the Operator Name
//
//  resetVisitedFlag
//
//  resets visited to FALSE
//
//  setVisitedFlag
//
//  sets visited to TRUE
//
//  getVisitedFlag
//
//  returns the value of visited (Boolean)
//
//  getVObjComponentsName
//
//  display the different components in the Operator
//
//  displayVersionNumber
//
//  Display the version number of the V_OBJECT to stdout
//
//  getVersionNumber
//
```

```
//  return the int versionNumber
//
//  dumpVObjSummary
//
//  dump predetermined attribute values to the stdout/stderr
//
//  setCreationDate
//
//  gets the system time and stores it in CreationDate
//
//  getCreationDate
//
//  Displays the creation date as a 26 character ascii text
//  string of the date and time
//
//  setLock
//
//  sets the lock to the system time
//
//  getWorker
//
//  returns the character string containing the workers name
//
//  getCreator
//
//  returns the character string containing the V_OBJECT
//  creators name
//
//  setWorker
//
//  gets the UNIX UserPtr variable and stores the value of that
//  variable into worker
//
//  resetLastOpTrue
//
//  Tells the system that the last operation on that V_OBJECT
//  was a checkin.  Prevents duplicate checkins from creating
//  new versioned objects on each checkin
//
//  resetLastOpFalse
//
//  resets the last operation immediately following checkin
//
//  get_last_operation
```

```
//
//  returns a Boolean TRUE if last operation was an add
//
//  releaseLock
//
//  sets the lockTime back to 0 (epoch time) Sometime in 1969.
//
//  getLockTime
//
//  return the lockTime as a time_t (long) structure
//
//  getDescription
//
//  Display the V_OBJECT description (if one exists)
//
//  listChildren
//
//  list V_OBJECT's chilren
//
//  longlistOperatorNames
//
//  list V_OBJECT's children
//
//  listOperatorNames
//
//  gives the explicit name of the operator (to include
//  path information from the root V_OBJECT
//
//  updateDescription
//
//  updates the versioned_objects description.
//
//  addCOMPONENTNode
//
//  adds an COMPONENT object to this V_OBJECT using the Ontos
//  binding mechanism
//
//  deleteChildNode
//
//  removes an operator from the children list of the
//  current V_OBJECT
//
//  addChildNode
//
```

```
//   adds an operator to the children list of the V_OBJECT
//
//   getParent
//
//   returns the parent V_OBJECT
//
//   getCOMPONENT
//
//   returns an COMPONENT pointer if one is contained in this
//   V_OBJECT
//
//   dumpSubtree
//
//   attempts to rebuild files from versioned objects onto
//   the UNIX subdirectory referenced by the UNIX variable
//   PROTOTYPE
//
//   releaseLockSubtree
//
//   resets that node and every node under it to zero
//   epoch time (sometime in 1969)
//
//   getChildren
//
//   returns a list of the children of the current V_OBJECT
//
//   getChildPtr
//
//   returns a Boolean TRUE if the Cardinality of the list
//   referenced by the theChildPtr is > zero
//
//   checkoutCOMPONENTNode
//
//   attempts to rebuild the .ps, .graph, .imp.psdl, .spec.psdl
//   and .a files of an operator/type stored in the Design
//   Database
//
// End -----------------------------------------------------

#endif
```

```
// File Header ------------------------------------------------------------
//.....:
//.Filename.....: versioned_object.cxx
// Date        : 9/16/91
// Author      : Garry Lewis
//             : Drew Dwyer
// Modified by  : Michael D. O'Loughlin
// Date        : 6/18/92
// Modifications : Extensive modifications have been conducted on almost all of the
/*              original code written by Dwyer and Lewis.  Every function that
                accesses an attribute of an object or the entire object has had
                to be modified. The original code is not even recognizable in
                some functions.
                Because the functionality of the design database system changed
                completely from what Dwyer and Lewis developed each individual
                modification has not been documented. Because of the massive
                changes required I discarded a lot of obsolete code and build a
                new system on the base Dwyer and Lewis had established.
*/
// Compiler     : Glockenspiel C++ 2.1
//
// End header comments ----------------------------------------------------


// SCCS ID follows: will compile to place date/time stamp in object file

static char versioned_object_cxx_SccsId[] = "@(#)versioned_object.cxx  1.3\t9/16/91";

// Contents ---------------------------------------------------------
//
//   V_OBJECT::V_OBJECT
//   V_OBJECT::V_OBJECT
//   V_OBJECT::Destroy
//   V_OBJECT::getDirectType
//   V_OBJECT::connect_vobject_to_thread
//   V_OBJECT::setParent
//   V_OBJECT::setNodeName
//   V_OBJECT::getNodeName
//   V_OBJECT::getVObjName
//   V_OBJECT::getName
//   V_OBJECT::resetVisitedFlag
//   V_OBJECT::setVisitedFlag
//   V_OBJECT::getVisitedFlag
//   V_OBJECT::getVObjComponentsName
```

```
//   V_OBJECT::displayVariationNumber
//   V_OBJECT::getVariationNumber
//   V_OBJECT::updateVariationNumber
//   V_OBJECT::displayVersionNumber
//   V_OBJECT::getVersionNumber
//   V_OBJECT::dumpVObjSummary
//   V_OBJECT::setCreationDate
//   V_OBJECT::getCreationDate
//   V_OBJECT::setLock
//   V_OBJECT::getWorker
//   V_OBJECT::getCreator
//   V_OBJECT::setWorker
//   V_OBJECT::resetLastOpTrue
//   V_OBJECT::resetLastOpFalse
//   V_OBJECT::get_last_operation
//   V_OBJECT::releaseLock
//   V_OBJECT::getLockTime
//   V_OBJECT::getDescription
//   V_OBJECT::listChildren
//   V_OBJECT::longlistOperatorNames
//   V_OBJECT::listOperatorNames
//   V_OBJECT::updateDescription
//   V_OBJECT::addCOMPONENTNode
//   V_OBJECT::deleteChildNode
//   V_OBJECT::addChildNode
//   V_OBJECT::getParent
//   V_OBJECT::getCOMPONENT
//   V_OBJECT::dumpSubtree
//   V_OBJECT::releaseLockSubtree
//   V_OBJECT::getChildren
//   V_OBJECT::getChildPtr
//   V_OBJECT::checkoutCOMPONENTNode
//
//   Description
//
//   methods for manipulating versioned objects
//
// End -----------------------------------------------------

 // Interface Requirements ---------------------------------------
//
// TURN ON TRACE-DEBUG

#define DEBUG
```

```
#include "debug.h"

#include <GlobalEntities.h>
#include <Directory.h>

#ifndef __VERSIONED_OBJECT_H
#include "versioned_object.h"
#endif

#ifndef __TRACER_H
#include "tracer.h"
#endif

#ifndef __DDBDEFINES_H
#include "ddbdefines.h"
#endif

#ifndef __VOBJECTFUNC_H
#include "vobjectfunc.h"
#endif
 // Constructor //

 // End Interface Requirements ---------------------------------

 extern Type *THREAD_OType;
 extern Type *V_OBJECT_OType;
 extern userPtr;
 extern char *dirNamePtr;


 V_OBJECT::V_OBJECT(APL *theAPL) : Object(theAPL)

    // Summary ----------------------------------------
    //
    //   This is an activation constructor required by ONTOS.
    //   ONTOS calls the activation constructor anytime an object
    //   is brought into memory. Note the constructor passes
    //   theAPL to the base class APL constructor.
    //
    // Parameter
    //
    // theAPL
    //
    //   A pointer to an APL (for Activation Parameter List) a
```

```
    //   structure.

{
 // empty by design
};

// End ----------------------------------------------------

// Constructor //

V_OBJECT::V_OBJECT(int versionNum)

    // Summary--------------------------------------------------
    //
    // Parameter
    //
    // Functional Description
    //
{
 initDirectType(V_OBJECT_OType);
 theVariationNumber = 0;
 theVersionNumber = versionNum;
 NumberOfVariations = 0;
 creationDate = setCreationDate();
 lockTime = 0;
 last_op_checkin = TRUE;
 visited = FALSE;
 created = TRUE;
 creator = new char [strlen(userPtr)+1];
 strcpy(creator,userPtr);
 worker = (char *)0;
 node_name = (char *)0;
 needsUpdating = FALSE;
 theDescriptionPtr.initToNull();
 theCOMPONENTPtr.initToNull();
 theThreadPtr.initToNull();
 theParentPtr.initToNull();

 List *newChildList = new List(V_OBJECT_OType);
 newChildList -> putObject();
 theChildPtr.Reset(newChildList, this);

 if (!THREAD_OType)
     {
```

```
        THREAD_OType = (Type*)OC_lookup("THREAD");
    }
 Dictionary *new_variation = new Dictionary(OC_integer,
                                THREAD_OType,
                                TRUE, FALSE);
 new_variation ->putObject();
 NewVariations.Reset(new_variation, this);

 Dictionary *variationPtr = new Dictionary(OC_integer,
                                V_OBJECT_OType,
                                TRUE, FALSE);
 variationPtr ->putObject();
 VariationList.Reset(variationPtr, this);

 putObject();
};

// End ----------------------------------------------------


// Member Function (in lieu of destructor)//

void V_OBJECT::Destroy(Boolean aborted)

   // Summary ------------------------------------------------
   //
   //  This one is semi tricky.  Destroy redefined the Destroy
   //  function inherited from the class CleanupObj.  Destroy()
   //  is used to delete CleanupObj objects and those of all its
   //  derived classes.  In defining any class that is directly
   //  or indirectly derived from CleanupObj, provide the
   //  function Destroy(Boolean aborted) in place of a destructor
   //  if there is any special processing required when the
   //  object's memory is deallocated.
   //
   // Parameter
   //
   //  aborted
   //
   //  If Destroy() is called as a result of an abort, aborted
   //  is set to TRUE; if it is called to delete the object for
   //  other reasons, aborted is set to False.
   //
   // Functional Description
```

```
    //
    //  CleanupObj in effect provides an audit trail of the
    //  creation of all stack-based instances of its derived
    //  classes, so that they can be cleanly deleted in the
    //  case of an abort during exception handling.  Hence
    //  the Destroy function.

{
 Object::Destroy(aborted);
};

// End ----------------------------------------------------

// Member Function //

Type* V_OBJECT::getDirectType()
{
 return V_OBJECT_OType;
}

// Member Function //

void V_OBJECT::connect_vobject_to_thread(THREAD *threadPtr)
{
 TRACER("V_OBJECT::connect_vobject_to_thread");
 theThreadPtr.Reset(threadPtr,this);
}

void V_OBJECT::setParent(V_OBJECT *parent)
{
 theParentPtr.Reset(parent,this);
}

void V_OBJECT::setNodeName(char *tree_node_name)
{
 char *token = (char *)0;
 token = strrchr(tree_node_name,'.');
 if (token)
   {
    node_name = new char[strlen(token)+1];
    strcpy(node_name,token);              // token in the subtree
    node_name++;                          // discard period (.)
   }
 else
```

```
  {
   node_name = new char [strlen(tree_node_name)+1];
   strcpy(node_name,tree_node_name);          // must be root root.
  }
}

char *V_OBJECT::getNodeName()
{
 return node_name;
}

char * V_OBJECT::getName()
{
 char *name;
 Directory *directory;


 if(!this)
  {
   cerr << "<ERROR: cannot get the name of a null V_OBJECT>\n";
   return NULL;
  }
 if (!theThreadPtr)
  {
   cerr << "<ERROR: cannot display the name of a null thread>\n";
   return NULL;
  }
 else
  {
   THREAD *myThreadPtr = (THREAD*) theThreadPtr.Binding(this);
   name =  myThreadPtr -> Name();
   OC_getNameComponents(name, &directory, &name);
   return name;
  }
}

// Member Function //

void V_OBJECT::getVObjName()
{
 char *name;
 Directory *directory;

 if(!this)
```

```
  {
   cerr << "<ERROR: cannot get the name of a null V_OBJECT>\n";
   return;
  }
 if (!theThreadPtr)
  {
   cerr << "<ERROR: cannot display the name of a null thread>\n";
   return;
  }
 else
  {
   THREAD *myThreadPtr = (THREAD*) theThreadPtr.Binding(this);
   name = myThreadPtr -> Name();
   OC_getNameComponents(name, &directory, &name);
   cout << name << "\n";
  }
}

// Member Function //

void V_OBJECT::getVObjComponentsName()
{ if(!this)
  {
   cerr << "<ERROR: cannot get the names of a null V_OBJECT>\n";
   return;
  }
 if (!theCOMPONENTPtr)
  {
   cerr << "<ERROR: This v_object does not have an COMPONENT component>\n";
   return;
  }
 else
  {
   COMPONENT *myCOMPONENTPtr = (COMPONENT*) theCOMPONENTPtr.Binding(this);
   myCOMPONENTPtr -> getComponentNames();
  }
}


// ************************************************************************

V_OBJECT *V_OBJECT::variation(int variationNumber)
    // Summary
    //
```

```
    //
    // End
    {
     TRACER ("V_OBJECT::variation");
     Dictionary *temp_list = (Dictionary*) VariationList.Binding(this);
     V_OBJECT *mytempvo = (V_OBJECT*) (Entity*) (*temp_list) [variationNumber];
     if (mytempvo) TRACE("mytempvo not NULL");
     return mytempvo;
    };

void V_OBJECT::displayVariationNumber()

   // Summary ----------------------------------------------------
   //
   //   This function displays the variation number of an object.

{
 cout << theVariationNumber << "  ";
};

int V_OBJECT::getVariationNumber()
{
 return theVariationNumber;
}

void V_OBJECT::updateVariationNumber(int NewVariationNumber)

   // Summary ----------------------------------------------------
   //
   //   This function updates the variation number of an object.

{
  theVariationNumber = NewVariationNumber;
};


void V_OBJECT::displayNumberOfVariations()

   // Summary ----------------------------------------------------
   //
   //   This function displays the number of variations of an object.

{
 cout << NumberOfVariations << "\n";
```

```
};

int V_OBJECT::getNumberOfVariations()
{
 return NumberOfVariations;
}

THREAD *V_OBJECT::getThread()
{
 if (theThreadPtr)
      {
       THREAD *tempThread = (THREAD *) theThreadPtr.Binding(this);
       return tempThread;
      }
 else
      {
       THREAD *nullthread = (THREAD *)0;
       return nullthread;
      }
}

void V_OBJECT::incrementNumberOfVariations()

   // Summary ---------------------------------------------------
   //
   //   This function updates the variation number of an object.

{
  NumberOfVariations = NumberOfVariations + 1;
};

void V_OBJECT::addVariation(V_OBJECT *V_ObjectPtr, int nextVariation)
{
 if (!this)
   {
    cerr << "<ERROR: cannot set the Variation node of a null V_OBJECT\n";
    return;
   }
 if (!V_ObjectPtr)
   {
    cerr << "<ERROR in AddVariation\n";
    return;
   }
  Dictionary *VarDictionaryPtr = (Dictionary*)VariationList.Binding(this);
```

```
 VarDictionaryPtr -> Insert(nextVariation, (Entity *) V_ObjectPtr);
 VarDictionaryPtr -> putObject();
}

void V_OBJECT::addVariationThread(THREAD *NewThreadPtr, int nextThread)
{
 if (!this)
   {
    cerr << "<ERROR: cannot set the Variation node of a null V_OBJECT\n";
    return;
   }
 if (!NewThreadPtr)
   {
    cerr << "<ERROR in Add Variation Thread.\n";
    return;
   }
 Dictionary *ThreadDictionaryPtr = (Dictionary*) NewVariations.Binding(this);
 ThreadDictionaryPtr -> Insert(nextThread, (Entity *) NewThreadPtr);
 ThreadDictionaryPtr -> putObject();
}

void V_OBJECT::copyChildren(V_OBJECT *child_list)
{

 if (!this)
   {
    cerr << "<ERROR: can not copy children of a null V_OBJECT!>\n";
    return;
   }

    List *child_copyfrom = (List *) child_list->theChildPtr.Binding(child_list);
    List *child_copyto = (List *) theChildPtr.Binding(this);
    ListIterator ChildrenPtr(child_copyfrom);
    V_OBJECT *cnode;

    while(ChildrenPtr.moreData())
      {
       cnode = (V_OBJECT *) (Entity *) ChildrenPtr();
       child_copyto->Insert(cnode);
      }
       child_copyto->putObject();
    return;
}
```

```
V_OBJECT *V_OBJECT::check_for_child(char *child_node_name)
{
 V_OBJECT *child = (V_OBJECT *)0;
 if (!this)
  {
   cerr << "<ERROR: no children for a null V_OBJECT!>\n";
   return child;
  }
   List *child_check_list = (List *) theChildPtr.Binding(this);
   ListIterator ChildrenPtr(child_check_list);
   while(ChildrenPtr.moreData())
    {
     child = (V_OBJECT *) (Entity *) ChildrenPtr();
     if (strcmp(child->getNodeName(),child_node_name) == 0)
      {
        return child;
      }
    }
   return  (V_OBJECT *)0;
}

Boolean V_OBJECT:: Needs_Updating()
    {
     return needsUpdating;
    }

void V_OBJECT::Update_Parent()
    {
     V_OBJECT *ParentPtr;
     ParentPtr = this->getParent();
     for (;;)
      if(ParentPtr !=NULL)
       {
        ParentPtr->needsUpdating = TRUE;
        ParentPtr = ParentPtr->getParent();
       }
      else
       {
        break;
       }
    }

void V_OBJECT::Reset_Update()
    {
```

```
    needsUpdating = FALSE;
    }

Boolean V_OBJECT::just_created()
    {
    return created;
    }

void V_OBJECT::set_created()
    {
    created = TRUE;
    }

void V_OBJECT::reset_created()
    {
    created = FALSE;
    }


// ***************************************************************************

// Member Function //

void V_OBJECT::displayVersionNumber()

   // Summary ---------------------------------------------------
   //
   //   This function displays the version number of an object.

{
 cout << theVersionNumber << "\n";
};

int V_OBJECT::getVersionNumber()
{
 return theVersionNumber;
}

// End -----------------------------------------------------


// Member Function //

void V_OBJECT::dumpVObjSummary()
```

```
{
 cout << ctime(&creationDate);
 cout << getCreator() << "\n";
 if (worker)
  {
   cout << worker << "\n";
  }
 else
  cout << "";

 if (!lockTime==0)
  cout << ctime(&lockTime);
 else
  cout << "\n\n";
  getDescription();
}

//Member Function //

time_t V_OBJECT::setCreationDate()
{
 time_t mytloc=0;
 time_t theTime;
 return theTime = time(mytloc);
}

//Member Function //

void V_OBJECT::setLock()
{
 lockTime = setCreationDate();
}

//Member function//

char *V_OBJECT::getWorker()
{
 return worker;
}

//Member function//

char *V_OBJECT::getCreator()
{
```

```
 return creator;
}

//Member function//

void V_OBJECT::setWorker()
{
 char *temp_worker = new char [strlen(userPtr)+1];
 strcpy(temp_worker,userPtr);
 if (worker)
   {
    delete worker;
   }
// commented out on 21 may 92 MO'L
// else
 worker = new char[strlen(temp_worker)+1];
 strcpy(worker,temp_worker);
}

//Member Function //

void V_OBJECT::resetVisitedFlag()
{
 visited = FALSE;
}

void V_OBJECT::setVisitedFlag()
{
 visited = TRUE;
}

Boolean V_OBJECT::getVisitedFlag()
{
 return visited;
}

//Member Function //

void V_OBJECT::resetLastOpTrue()
{
 last_op_checkin = TRUE;
}

void V_OBJECT::resetLastOpFalse()
```

```
{
 last_op_checkin = FALSE;
}

//Member Function //

Boolean V_OBJECT::get_last_operation()
{
 return last_op_checkin;
}

//Member Function //

int V_OBJECT::releaseLock()
{
 if (!worker)
   {
    last_op_checkin = TRUE;
    lockTime = 0;
    return SUCCESS;
   }

 if (strcmp(userPtr,getWorker())==0)
   {
    last_op_checkin = TRUE;
    lockTime = 0;
    delete worker;
    worker = (char *)0;
    return SUCCESS;
   }
 else
   {
    cerr << "<ERROR: Only " << getWorker() << " May unlock this object!...Aborting>\n";
    return FAILED;
   }
}

//Member Function //

time_t V_OBJECT::getCreationDate()
{
 return creationDate;
}
```

```
// Member Function //

time_t V_OBJECT::getLockTime()
{
 return lockTime;
}

// Member Function //

void V_OBJECT::getDescription()

   // Summary --------------------------------------------------
   //
   //   This function displays the description of an object
   //

{
 if(!this)
   {
    cout << "<ERROR: cannot get the description of a null V_OBJECT>\n";
    return;
   }
 if (!theDescriptionPtr)
   {
    cerr << "<ERROR: This v_object does not have a description>\n";
    return;
   }
 else
   {

    TEXT_OBJECT *myTextObjectPtr =
     (TEXT_OBJECT*) theDescriptionPtr.Binding(this);
    myTextObjectPtr ->text(cout);
   }
}

// Member Function //

void V_OBJECT::updateDescription(char *fileName, ifstream& input_file_stream)
{
 if(!this)
   {
    cerr << "<ERROR: cannot update the description of a null V_OBJECT>\n";
    return;
```

```
  }

 else
  {
   if (strcmp(userPtr,getCreator())==0)
    {
     if(!theDescriptionPtr)
      {
       TEXT_OBJECT *textObjectPtr = new TEXT_OBJECT();
       textObjectPtr -> append(fileName, input_file_stream);
       textObjectPtr ->putObject();
       theDescriptionPtr.Reset(textObjectPtr, this);
      }
     else
      {
       TEXT_OBJECT *textObjectPtr =
        (TEXT_OBJECT*) theDescriptionPtr.Binding(this);
       textObjectPtr -> resetTheText();
       textObjectPtr -> append(fileName, input_file_stream);
      }
    }
   else
    cerr << "<ERROR: only " << getCreator() << " may update description.  Aborting...>\n";
  }
 putObject();
}

// Member Function //

void V_OBJECT::addCOMPONENTNode(COMPONENT *myCOMPONENTPtr)
{
 if (!this)
  {
   cerr << "<ERROR: cannot set the COMPONENT node of a null V_OBJECT\n";
   return;
  }
 if (!myCOMPONENTPtr)
  {
   cerr << "<ERROR: cannot give to a V_OBJECT a null COMPONENT>\n";
   return;
  }
 theCOMPONENTPtr.Reset(myCOMPONENTPtr, this);
}
```

```
// Member Function //

void V_OBJECT::deleteChildNode(V_OBJECT *myV_ObjPtr)
{
 List *child_nodes = (List *)theChildPtr.Binding(this);

 if (!this)
   {
    cerr << "<ERROR: cannot delete the child node of a null V_OBJECT\n";
    return;
   }
 if (!child_nodes)
   {
    cerr << "<ERROR: cannot remove a NULL child>\n";
    return;
   }
 long location =0;
 if (child_nodes->isMember(myV_ObjPtr))
   {
    location = child_nodes->Index(myV_ObjPtr);
    child_nodes->Remove(location);
   }
}

// Member Function //

void V_OBJECT::addChildNode(V_OBJECT *myV_ObjPtr)
{
 List *child_nodes = (List *)theChildPtr.Binding(this);

 if (!this)
   {
    cerr << "<ERROR: cannot set the child node of a null V_OBJECT\n";
    return;
   }
 if (!child_nodes)
   {
    cerr << "<ERROR: cannot give to a V_OBJECT a null child>\n";
    return;
   }
 child_nodes->Insert(myV_ObjPtr);
}

//Member Function //
```

```
V_OBJECT *V_OBJECT::getParent()
{
 return (V_OBJECT *)(Entity *) theParentPtr.Binding(this);
}

COMPONENT *V_OBJECT::getCOMPONENT()
{
 return (COMPONENT *) (Entity *) theCOMPONENTPtr.Binding(this);
}

void V_OBJECT::listChildren()
{

 if (!this)
   {
    cerr << "<ERROR: can not dump children of a null V_OBJECT!>\n";
    return;
   }

 if(!theChildPtr)
   {
    cerr << "<ERROR: can not print children of a null childPtr!>\n";
    return;
   }
 else
   {

    List *child_nodes = (List *) theChildPtr.Binding(this);
    ListIterator ChildrenPtr(child_nodes);
    V_OBJECT *cnode;

    while(ChildrenPtr.moreData())
     {
      cnode = (V_OBJECT *) (Entity *) ChildrenPtr();
      cout << cnode->getNodeName();
      //************
      // following for loop provides spacing...
      // ***********
      int i=0;
      for (i=0;i<(PRINT_VERSION_LOCATION-strlen(cnode->getNodeName()));i++)
       cout << " ";
      cout << cnode->getVariationNumber();
      cout << " ";
      cout << cnode->getVersionNumber();
```

```
        cout << "\n";
       }
     return;
    }
}

void V_OBJECT::longlistOperatorNames()
{
 TRACER("V_OBJECT::longlistOperatorNames");
 if (!this)
   {
    cerr << "<ERROR: can not dump operators of a null V_OBJECT!>\n";
    return;
   }

 if(!theChildPtr)
   {
    cerr << "<ERROR: can not print list of a null childPtr!>\n";
    return;
   }
 else
   {

    List *child_nodes = (List *) theChildPtr.Binding(this);
    ListIterator ChildrenPtr(child_nodes);
    V_OBJECT *cnode;
    int line_feed = 0;
    while(ChildrenPtr.moreData())
     {
      TRACE("iteration in while loop");
      cnode = (V_OBJECT *) (Entity *) ChildrenPtr();
      //     if (cnode->getChildPtr())
      //      {
      //        cnode-> longlistOperatorNames();
      //      }
// 1-23-92       cerr << "Operator:    ";
      if (line_feed) cout << "\n" ;  // removes blank line from output
      TRACE(cnode->getNodeName() );
      cout << cnode->getNodeName() ;
      line_feed++;
      //************
      // following for loopprovides spacing...
      // ***********
      int i=0;
```

```
        for (i=0;i<(PRINT_VERSION_LOCATION-strlen(cnode->getNodeName()));i++)
          cout << " ";
        cout << cnode->getVariationNumber();
         cout << " ";
        cout << cnode->getVersionNumber();
        cout << " ";
        time_t locktime = cnode->getLockTime();
       }
      cout << "\n";
    }
}

void V_OBJECT::listOperatorNames()
{
 Boolean node_was_visited = FALSE;
 if (!this)
   {
    cerr << "<ERROR: can not dump children of a null V_OBJECT!>\n";
    return;
   }
 if(!theChildPtr)
   {
    cerr << "<ERROR: can not print children of a null childPtr!>\n";
    return;
   }
 else
   {
    List *child_nodes = (List *) theChildPtr.Binding(this);
    ListIterator ChildrenPtr(child_nodes);
    V_OBJECT *cnode;
    int index = 1;
    while(ChildrenPtr.moreData())
     {
      cnode = (V_OBJECT *) (Entity *) ChildrenPtr();
      node_was_visited = cnode->getVisitedFlag();
      if (cnode->getChildPtr())
        {
         cnode-> listOperatorNames();
        }
      if (node_was_visited && index > 1)
        {
         cnode->resetVisitedFlag();
         cnode->putObject();
         break;
```

```
        }
      else
        {
         cnode->resetVisitedFlag();
         cnode->putObject();
        }
      if (index ==1)
       cnode->setVisitedFlag();
      index++;
      char *name;
      name = cnode->getName();
      name[strlen(name)-2] = '\0';
      cout << name;
      //************
      // following for loop provides spacing...
      // ***********
      //     int i=0;
      //     for (i=0;i<(PRINT_VERSION_LOCATION-strlen(cnode->getName()));i++)
      //         cout << " ";
      cout << " ";
      cout << cnode->getVariationNumber();
      cout << " ";
      cout << cnode->getVersionNumber();
      cout << "\n";
      time_t locktime = cnode->getLockTime();
      cout << ctime(&locktime);
     }
    resetVisitedFlag();
    putObject();
    return;
   }
}

void V_OBJECT::releaseLockSubtree()
{
 List *child_nodes = (List *) theChildPtr.Binding(this);
 ListIterator ChildrenPtr(child_nodes);
 V_OBJECT *cnode;

 while(ChildrenPtr.moreData())
  {
   cnode = (V_OBJECT *) (Entity *) ChildrenPtr();
   if (cnode->releaseLock())
    cnode->putObject();
```

```
    else
      {
       cerr << "<ERROR: while unlocking " << cnode->getName() << " Aborting...>\n";
       break;   // should try to unlock other siblings and their children.
      }
    if (cnode->getChildPtr())
      {
       cnode-> releaseLockSubtree();
      }
   }
 return;
}


void V_OBJECT::dumpSubtree(char *file_write_option)
{
 TRACER("V_OBJECT::dumpSubtree");
 Boolean node_was_visited = FALSE;
 Boolean file_operation_successful = FALSE;
 Boolean write_operation = FALSE;
 if (strcmp(file_write_option,"w")==0 || strcmp(file_write_option,"W")==0)
  write_operation = TRUE;
 if (!this)
  {
   cerr << "<ERROR: can not dump children of a null V_OBJECT!>\n";
   return;
  }
 if(!theChildPtr)
  {
   cerr << "<ERROR: can not print children of a null childPtr!>\n";
   return;
  }
 else
  {
    if (write_operation)
      {
        DDBControlData VControlFile;
        DDBControlData *VControlFilePtr = &VControlFile;
        char *Vnew_name = NULL;
        char *VV_name = getName();
        VV_name [strlen(VV_name) - 2] = '\0';
        char *VDDBControlpath = new char [strlen(dirNamePtr) + strlen(VV_name) + 13];
        strcpy(VDDBControlpath,dirNamePtr);
        strcat(VDDBControlpath, "/");
```

```
        strcat(VDDBControlpath, "ddbCtrlData.");
        strcat(VDDBControlpath, VV_name);
        VControlFilePtr->variation = getVariationNumber();
        VControlFilePtr->version = getVersionNumber();
        putddbControlFile(VDDBControlpath,VControlFilePtr);
      }
List *child_nodes = (List *) theChildPtr.Binding(this);
ListIterator ChildrenPtr(child_nodes);
V_OBJECT *cnode;
int index = 1;
while (cnode = (V_OBJECT *) (Entity *)ChildrenPtr())
 {
   node_was_visited = cnode->getVisitedFlag();
   if (cnode->getChildPtr())
     {
      cnode-> dumpSubtree(file_write_option);
     }
   if (node_was_visited && index > 1)
     {
      cnode->resetVisitedFlag();
      cnode->putObject();
      break;
     }
   else
     {
      cnode->resetVisitedFlag();
      cnode->putObject();
     }
   if (index ==1)
    cnode->setVisitedFlag();
   else
     {
      cnode->resetVisitedFlag();
      cnode->putObject();
     }
   index++;
   long cobject_locktime = 0;
   cobject_locktime = cnode->getLockTime();
   if (cobject_locktime>0)              // prevent checkout
     {
      if (strcmp(file_write_option,"w")==0 )
        {
         cerr << "<ERROR:Module " << cnode->getNodeName()
           << " lockedby " << cnode->getWorker()
```

```
                << " Reset write option to display>\n";
              strcpy(file_write_option,"r");
             }
           cerr << "<Caution: " << cnode->getNodeName()
             << " locked on: "
              << ctime(&cobject_locktime)
               << "Node checked out for display only\n";
         }
        file_operation_successful = cnode->checkoutCOMPONENTNode(file_write_option);
        if ((file_operation_successful) &&
                ((strcmp(file_write_option,"W")==0) ||
                 (strcmp(file_write_option,"w")==0)))
         {
          cnode->setLock();
          cnode->setWorker();
          cnode->resetLastOpFalse();
          cnode->putObject();

          DDBControlData ControlFile;
          DDBControlData *ControlFilePtr = &ControlFile;
          char *new_name = NULL;
          char *V_name = cnode->getName();
          V_name [strlen(V_name) - 2] = '\0';
          char *DDBControlpath = new char [strlen(dirNamePtr) + strlen(V_name) + 13];
          strcpy(DDBControlpath,dirNamePtr);
          strcat(DDBControlpath, "/");
          strcat(DDBControlpath, "ddbCtrlData.");
          strcat(DDBControlpath, V_name);
          ControlFilePtr->variation = cnode-> getVariationNumber();
          ControlFilePtr->version = cnode-> getVersionNumber();
          putddbControlFile(DDBControlpath,ControlFilePtr);

         }
        if (write_operation)
          strcpy(file_write_option,"w");
       }
      resetVisitedFlag();
      putObject();
      return;
    }
}

List * V_OBJECT::getChildren()
{
```

```
 List *temp_list = (List *)theChildPtr.Binding(this);
 if (temp_list->Cardinality() > 0)
  return temp_list;
 else
  return NULL;
}

Boolean V_OBJECT::getChildPtr()
{
 List *temp_list = (List *)theChildPtr.Binding(this);
 if (temp_list->Cardinality() > 0)
  return TRUE;
 else
  return FALSE;
}

// Member Function //

Boolean V_OBJECT::checkoutCOMPONENTNode(char *file_write_option)
{
 TRACER("V_OBJECT::checkoutCOMPONENTNode");
 if (!this)
   {
    cerr << "<ERROR: cannot checkout COMPONENT nodes of a null V_OBJECT\n";
    return FAILED;
   }
 if (!theCOMPONENTPtr)
   {
    cerr << "<ERROR: cannot get the source code from a null COMPONENT>\n";
    return FAILED;
   }
 else
   {
    COMPONENT *myCOMPONENTPtr = (COMPONENT*) theCOMPONENTPtr.Binding(this);
    return (myCOMPONENTPtr -> getComponentSource(file_write_option));
   }
}
```

```
// File Header ------------------------------------------------------------
 //.............:
 //.Filename.....: vobjectfunc.h
// Date        : 9/16/91
// Author      : Garry Lewis
//             : Drew Dwyer
// Modified by  : Michael D. O'Loughlin
// Date        : 6/18/92
// Modifications : Extensive modifications have been conducted on almost all of the
/*              original code written by Dwyer and Lewis.  Every function that
                accesses an attribute of an object or the entire object has had
                to be modified. The original code is not even recognizable in
                some functions.
                Because the functionality of the design database system changed
                completely from what Dwyer and Lewis developed each individual
                modification has not been documented. Because of the massive
                changes required I discarded a lot of obsolete code and build a
                new system on the base Dwyer and Lewis had established.
*/
// Compiler     : Glockenspiel C++ 2.1
//
// End header comments ----------------------------------------------------



#ifndef __VOBJECTFUNC_H
#define __VOBJECTFUNC_H

#ifndef __DDBDEFINES_H
#include "ddbdefines.h"
#endif

#ifndef __THREAD_H
#include "thread.h"
#endif
 // SCCS ID follows: will compile to place date/time stamp in object file

 static char vobjectfunc_h_SccsId[] = "@(#)vobjectfunc.h      1.3\t9/16/91";

// Contents ---------------------------------------------------
 //
 //   VOBJECTFUNC
 //
 // Description
```

```
//
//   Defines functions manipulating Versioned Objects
//   (Operators) as called by main()
//
// End -----------------------------------------------------

// Interface Dependencies ----------------------------------
//
// NONE
//
// End Interface Dependencies ------------------------------

void list_operators_func(int, char *, char *, char *, char *);
void update_vobject_desc_func(int, char *,char *,char *,char *,char *);
void get_vobject_desc_func(int ,char *, char *,char *,char *);
void get_vobject_date_func(int , char *,char  *,char *,char *);
void get_vobject_versions_func(int , char *,char *);
void get_vobject_lock_func(int, char *, char *, char *,char *);
void get_vobject_version_func();
void dump_vobject_summary_func(int , char *,char  *,char *,char *);
void get_vobject_psfile_func(int , char *, char *,char *, char *);
void get_vobject_graphfile_func(int , char *,char  *,char *, char *);
void get_vobject_impfile_func(int , char *, char  *,char *, char *);
void get_vobject_specfile_func(int , char *,char  *,char *, char *);
void get_vobject_sourcefile_func(int , char *,char  *,char *, char *);
void dump_vobject_files_func(int , char *, char *,char *, char *,char *);
void dump_vobject_tree_func(int , char *, char *,char *, char *, char *);
void long_list_children_func(int,char *, char *, char *);
void long_list_parents_func(int,char *, char *, char *, char *);
void long_list_operators_func(int, char *, char *, char *, char *);
void release_operator_lock_func(int, char *, char *, char *, char *);
void release_subtree_lock_func(int, char *, char *, char *, char *);
//void add_vobject_and_subtree_func(int , char *, char *);
void add_new_variation_func(int , char *, char *, char *, char *);
void historical_trail(int, char *, char *, char *, char *);
void putddbControlFile(char*, struct DDBControlData*);
DDBControlData* getddbControlFile(char *);
THREAD* firstThread(THREAD*, int);
V_OBJECT* getDefaultVObject(THREAD*);
THREAD* findThread(char*, char*);

// Description ---------------------------------------------
//
//   list_operators_func
```

```
//
//  Provides a list of operators associated with a
//  subtree of a designated versioned_object.
//
//  update_vobject_desc_func
//
//  Updates the description text of a designated versioned
//  object.
//
//  get_vobject_desc_func
//
//  Provides a description of the designated versioned object.
//
//  get_vobject_date_func
//
//  Displays the date that the versioned object was created.
//
//  get_vobject_versions_func
//
//  Checks the thread and displays the different versions.
//
//  get_vobject_lock_func
//
//  Displays a 26 character text entry reflecting the time
//  and date that the versioned object was locked.
//
//  get_vobject_version_func
//
//  returns the version of the versioned object instance.
//
//  dump_vobject_summary_func
//
//  displays predetermined attribute fields to stdout/stderr.
//
//  get_vobject_psfile_func
//
//  rebuilds the CAPS postscript file to the PROTOTYPE
//  directory.
//
//  get_vobject_graphfile_func
//
//  rebuilds the CAPS graph file to the PROTOTYPE directory.
//
//  get_vobject_impfile_func
```

```
//
//  rebuilds the CAPS implementation file to the PROTOTYPE directory.
//
//  get_vobject_specfile_func
//
//  rebuilds the CAPS specification file to the PROTOTYPE directory.
//
//  get_vobject_sourcefile_func
//
//  rebuilds the CAPS source file to the PROTOTYPE directory.
//
//  dump_vobject_files_func
//
//  rebuilds all of the above files (of one operator) to the
//  PROTOTYPE directory.
//
//  dump_vobject_tree_func
//
//  rebuilds all existing TEXT_OBJECT attributes to the PROTOTYPE
//  directory for the entire decomposition of the root operator
//  down to the component operators.  May be dumped in "read only"
//  or "read/write".  When dumped, all versioned objects are
//  locked for modification by other users.
//
//  long_list_children_func
//
//  lists the node name and version number of children.
//
//  long_list_parents_func
//
//  lists the most current parent (and that parents siblings).
//
//  long_list_operators_func
//
//  lists the node name and version number of a node's children.
//
//  release_operator_lock_func
//
//  reset the locktime of a versioned operator.
//
//  release_subtree_lock_func
//
//  reset the locktime of a versioned operator and all children
//  of that versioned operator.
```

```
//
//   add_vobject_and_subtree_func
//
//   the reverse of a dumping vobjects. this checks versioned
//   objects back into the database, versioning them if necessary.
//
// End Description ----------------------------------------------

#endif  // end vobjectfunc header function
```

```
// File Header ------------------------------------------------------------
 //.............:
 //.Filename.....: vobjectfunc.cxx
// Date        : 9/16/91
// Author      : Garry Lewis
//             : Drew Dwyer
// Modified by  : Michael D. O'Loughlin
// Date        : 6/18/92
// Modifications : Extensive modifications have been conducted on almost all of the
/*              original code written by Dwyer and Lewis.  Every function that
                accesses an attribute of an object or the entire object has had
                to be modified. The original code is not even recognizable in
                some functions.
                Because the functionality of the design database system changed
                completely from what Dwyer and Lewis developed each individual
                modification has not been documented. Because of the massive
                changes required I discarded a lot of obsolete code and build a
                new system on the base Dwyer and Lewis had established.
*/
// Compiler     : Glockenspiel C++ 2.1
//
// End header comments ----------------------------------------------------

// SCCS ID follows: will compile to place date/time stamp in object file

static char vobjectfunc_cxx_SccsId[] = "@(#)vobjectfunc.cxx     1.3\t9/16/91";

// Contents ---------------------------------------------------------
 //
 //   list_operators_func
 //   update_vobject_desc_func
 //   get_vobject_desc_func
 //   get_vobject_date_func
 //   get_vobject_versions_func
 //   get_vobject_lock_func
 //   get_vobject_version_func
 //   dump_vobject_summary_func
 //   get_vobject_psfile_func
 //   get_vobject_graphfile_func
 //   get_vobject_impfile_func
 //   get_vobject_specfile_func
 //   get_vobject_sourcefile_func
 //   dump_vobject_files_func
 //   dump_vobject_tree_func
```

```
//    long_list_children_func
//    long_list_parents_func
//    long_list_operators_func
//    release_operator_lock_func
//    release_subtree_lock_func
//    add_vobject_and_subtree_func
//
// Description
//
// this contains the functions for manipulating versioned objects
//
// End -----------------------------------------------------------

// Interface Dependencies ---------------------------------------
//

// TURN ON TRACE-DEBUG
#define DEBUG
#include "debug.h"

#include "My_String.h"

#include <stream.hxx>
#include <List.h>
#include <Directory.h>

extern "C--"
{
#include <stdlib.h>
}

#ifndef __DIRECTORY_H
#include "directory.h"
#endif

#ifndef __TREE_H
#include "tree.h"
#endif

#ifndef __TREENODE_H
#include "treenode.h"
#endif

#ifndef __PROTOTYPE_H
```

```
#include "prototype.h"
#endif

#ifndef __COMPONENT_H
#include "component.h"
#endif

#ifndef __VOBJECT_H
#include "versioned_object.h"
#endif

#ifndef __THREAD_H
#include "thread.h"
#endif

#ifndef __VOBJECTFUNC_H
#include "vobjectfunc.h"
#endif

#ifndef __DDBDEFINES_H
#include "ddbdefines.h"
#endif

//
// End Interface Dependencies ----------------------------------


extern Type *THREAD_OType;
extern Directory *prototype_dir;
extern char *dirNamePtr;
PROTOTYPE *prototypeptr;
THREAD *threadPtr;
COMPONENT *COMPONENTPtr;
long vobject_locktime =0;


void list_operators_func(int number_arguments, char *proto_name,
                char *operator_name, char * variationstr, char *versionstr)
{
  TRACER("list_operators_func");

  V_OBJECT *vobjectPtr;
  threadPtr = findThread(proto_name, operator_name);
  if (!threadPtr)
```

```
  {
   return;
  }

 switch (number_arguments)
  {
  case 2:
    {
     vobjectPtr = getDefaultVObject(threadPtr);
     vobjectPtr ->listOperatorNames();
     break;
    }
  case 4:
    {
     int varNum = atoi(variationstr);
     threadPtr = firstThread (threadPtr, varNum);
     V_OBJECT *vobjectPtr;
     vobjectPtr = threadPtr->version(atoi(versionstr));
     vobjectPtr ->listOperatorNames();
     break;
    }
  default:
    cerr << "<ERROR: invalid number args for get vobject description>\n";
  }
}

void update_vobject_desc_func(int number_arguments,char *proto_name,
                    char *operator_name, char *filename,
                    char *variationstr, char *versionstr)
{
  TRACER("update_vobject_desc_func");
  V_OBJECT *vobjectPtr;
  threadPtr = findThread(proto_name, operator_name);
  if (!threadPtr)
  {
   return;
  }

 ifstream description_file;
 description_file.open(filename, ios :: in);

 if (!description_file)
  {
   cout << "<Description File does not exist! >\n";
```

```
  }
 else
  {
   switch (number_arguments)
     {
      case 5:
       int varNum = atoi(variationstr);
       threadPtr = firstThread (threadPtr,varNum);
       vobjectPtr = threadPtr->version(atoi(versionstr));
       vobjectPtr ->updateDescription(filename,description_file);
       break;
      default:
       cerr << "<ERROR: invalid number args for update vobject description>\n";
     }
   }
}

void get_vobject_desc_func(int number_arguments,char *proto_name,
                    char *operator_name, char *variationstr,
                       char *versionstr)
{
 TRACER("get_vobject_desc_func");
 V_OBJECT *vobjectPtr;
 threadPtr = findThread(proto_name, operator_name);
 if (!threadPtr)
  {
   return;
  }
 switch (number_arguments)
   {
   case 2:
   {
    V_OBJECT *vobjectPtr;
    vobjectPtr = threadPtr->current();
    vobjectPtr ->getDescription();
    break;
   }
   case 4:
   {
    int varNum = atoi(variationstr);
    threadPtr = firstThread (threadPtr, varNum);
    vobjectPtr = threadPtr->version(atoi(versionstr));
    vobjectPtr ->getDescription();
    break;
```

```
  }
  default:
   cerr << "<ERROR: invalid number args for get vobject description>\n";
  }
}

void release_subtree_lock_func(int number_arguments, char *proto_name,
                    char *operator_name, char *variationstr,
                     char *versionstr)
{
 TRACER("release_subtree_lock_func");
 V_OBJECT *vobjectPtr;
 threadPtr = findThread(proto_name, operator_name);
 if (!threadPtr)
  {
   return;
  }

 switch (number_arguments)
  {
  case 2:
   cerr << "VRS operation no longer valid with default version number."
        << " Variation & version number required.";
   break;
  case 4:
   {
    int varNum = atoi(variationstr);
    threadPtr = firstThread (threadPtr, varNum);
    vobjectPtr = threadPtr->version(atoi(versionstr));
    if (vobjectPtr ->releaseLock())
     {
      vobjectPtr->putObject();
      vobjectPtr ->releaseLockSubtree();
      cerr << "Subtree locks released.\n";
     }
    else
     {
     cerr << "<ERROR: Can't unlock "<<vobjectPtr->getName()
          << " Abort releaseLock rest subtree>\n";
     }
    break;
   }
  default:
   cerr << "<ERROR: invalid number args for subtree release lock>\n";
```

```
  }
}

void release_operator_lock_func(int number_arguments, char *proto_name,
                    char *operator_name, char *variationstr,
                    char *versionstr)
{
 TRACER("release_operator_lock_func");
 V_OBJECT *vobjectPtr;
 threadPtr = findThread(proto_name, operator_name);
 if (!threadPtr)
  {
   return;
  }

 switch (number_arguments)
  {
  case 2:
   cerr << "VRO operation no longer valid with default version number."
        << " Variation & version number required.";
   break;
  case 4:
   {
    int varNum = atoi(variationstr);
    threadPtr = firstThread (threadPtr, varNum);
    vobjectPtr = threadPtr->version(atoi(versionstr));
    if (vobjectPtr->releaseLock())
     {
      vobjectPtr->putObject();
     }
    else
     {
    cerr << "<ERROR: Can't unlock "<<vobjectPtr->getName()
         << " Aborting release lock>\n";
     }
    break;
   }
  default:
   cerr << "<ERROR: invalid number args for release lock>\n";
  }
}

void get_vobject_date_func(int number_arguments, char *proto_name,
                char *operator_name, char *variationstr,
```

```
                    char *versionstr)
{
 TRACER("GET_vobject_date_func");
 time_t creation_date;
 V_OBJECT *vobjectPtr;
 threadPtr = findThread(proto_name, operator_name);
 if (!threadPtr)
  {
   return;
  }

 switch (number_arguments)
  {
  case 2:
    vobjectPtr = getDefaultVObject(threadPtr);
    creation_date = vobjectPtr ->getCreationDate();
    cout << ctime(&creation_date);
    break;
  case 4:
    int varNum = atoi(variationstr);
    threadPtr = firstThread (threadPtr, varNum);
    vobjectPtr = threadPtr->version(atoi(versionstr));
    creation_date = vobjectPtr ->getCreationDate();
    cout << ctime(&creation_date) ;
    break;
  default:
    cerr << "<ERROR: invalid number args for get vobject date>\n";
  }
}

void get_vobject_versions_func(int number_arguments, char *proto_name,
                    char *operator_name)

{
 TRACER("GET_vobject_versions_func");
 threadPtr = findThread(proto_name, operator_name);
 if (!threadPtr)
  {
   return;
  }

 switch (number_arguments)
  {
  case 2:
```

```
    V_OBJECT *initialVersion = threadPtr-> version(1);
    int count = initialVersion->getNumberOfVariations();
    for (int i=1; i <= count; i++)
      {
       THREAD *TthreadPtr = firstThread (threadPtr, i);
       TthreadPtr->displayThreadVersions();
      }
    break;
   default:
    cerr << "<ERROR: invalid number args for get vobject VERSIONS>\n";
   }
}

void historical_trail(int number_arguments, char *proto_name,
                      char *operator_name, char *variationstr,
                         char *versionstr)
{
 V_OBJECT *has_parent;
 THREAD *previousThreadPtr = (THREAD *)0;
 threadPtr = findThread(proto_name, operator_name);
 if (!threadPtr)
  {
   return;
  }

 previousThreadPtr = firstThread (threadPtr, atoi(variationstr));
 has_parent = previousThreadPtr->version(atoi(versionstr));

 char *name;
 name = has_parent->getName();
 name[strlen(name)-2] = '\0';
 cout << name << " " << has_parent->getVariationNumber() << " "
      << has_parent->getVersionNumber() << "\n";

 while (!(previousThreadPtr->previousVariation() == 0))
  {
   cout << name << " " << previousThreadPtr->previousVariation() << " "
     << previousThreadPtr->previousVersion() << "\n";
   previousThreadPtr =
     firstThread (threadPtr, previousThreadPtr->previousVariation());
  }
}
```

```
void get_vobject_lock_func(int number_arguments, char *proto_name,
                    char *operator_name, char *variationstr,
                       char *versionstr)
{
 TRACER("GET_vobject_lock_func");
 time_t lock_time;
 V_OBJECT *vobjectPtr;
 threadPtr = findThread(proto_name, operator_name);
 if (!threadPtr)
  {
   return;
  }

 switch (number_arguments)
  {
  case 2:
    vobjectPtr = getDefaultVObject(threadPtr);
    vobjectPtr = threadPtr->current();
    lock_time = vobjectPtr ->getLockTime();
    cout << ctime(&lock_time);
    break;
  case 4:
    threadPtr = firstThread (threadPtr, atoi(variationstr));
    vobjectPtr = threadPtr->version(atoi(versionstr));
    lock_time = vobjectPtr ->getLockTime();
    cout << ctime(&lock_time);
    break;
  default:
    cerr << "<ERROR: invalid number args for get vobject lock>\n";
   }
}

void get_vobject_version_func()
{
 cerr << "Not implemented.  Unclear specs for get version of vobject\n";
}

void dump_vobject_summary_func(int number_arguments, char *proto_name,
                      char *operator_name, char *variationstr,
                       char *versionstr)
{
 TRACER("dump_vobject_summary_func");
 V_OBJECT *vobjectPtr;
 threadPtr = findThread(proto_name, operator_name);
```

```
 if (!threadPtr)
  {
   return;
  }

 switch (number_arguments)
   {
  case 2:
    vobjectPtr = getDefaultVObject(threadPtr);
    vobjectPtr->dumpVObjSummary();
    break;
  case 4:
    threadPtr = firstThread (threadPtr, atoi(variationstr));
    vobjectPtr = threadPtr->version(atoi(versionstr));
    vobjectPtr->dumpVObjSummary();
    break;
  default:
    cerr << "<ERROR: invalid number args for get vobject summary>\n";
   }
}

void get_vobject_psfile_func(int number_arguments, char *proto_name,
                    char *operator_name,
                      char *variationstr, char *versionstr)
{
 TRACER("GET_vobject_psfile_func");
 V_OBJECT *vobjectPtr;
 threadPtr = findThread(proto_name, operator_name);
 if (!threadPtr)
  {
   return;
  }

 switch (number_arguments)
  {
  case 2:
   {
    vobjectPtr = getDefaultVObject(threadPtr);
    vobject_locktime = vobjectPtr->getLockTime();
    COMPONENTPtr =vobjectPtr->getCOMPONENT();
    COMPONENTPtr->getPSfile("R");
    break;
   }
  case 4:
```

```
    {
     threadPtr = firstThread (threadPtr, atoi(variationstr));
     vobjectPtr = threadPtr->version(atoi(versionstr));
     char *file_write_option = "R";
     COMPONENTPtr = vobjectPtr->getCOMPONENT();
     COMPONENTPtr ->getPSfile(file_write_option);
     break;
    }
   default:
     cerr << "<ERROR: invalid number args for get vobject PS>\n";
  }
}


void get_vobject_graphfile_func(int number_arguments, char *proto_name,
                        char *operator_name,
                        char *variationstr, char *versionstr)
{
 TRACER("GET_vobject_graphfile_func");
 V_OBJECT *vobjectPtr;
 threadPtr = findThread(proto_name, operator_name);
 if (!threadPtr)
  {
   return;
  }

 switch (number_arguments)
  {
  case 2:
     {
      vobjectPtr = getDefaultVObject(threadPtr);
      COMPONENTPtr =vobjectPtr->getCOMPONENT();
      COMPONENTPtr->getGRAPHfile("R");
      break;
     }
  case 4:
     {
      threadPtr = firstThread (threadPtr, atoi(variationstr));
      V_OBJECT *vobjectPtr;
      vobjectPtr = threadPtr->version(atoi(versionstr));
      COMPONENTPtr = vobjectPtr->getCOMPONENT();
      COMPONENTPtr ->getGRAPHfile("R");
    break;
     }
```

```
  default:
    cerr << "<ERROR: invalid number args for get vobject GRAPH>\n";
  }
}

void get_vobject_impfile_func(int number_arguments, char *proto_name,
                    char *operator_name,char *variationstr,
                      char *versionstr)
{
 TRACER("GET_vobject_impfile_func");
 V_OBJECT *vobjectPtr;
 threadPtr = findThread(proto_name, operator_name);
 if (!threadPtr)
  {
   return;
  }

 switch (number_arguments)
  {
  case 2:
    vobjectPtr = getDefaultVObject(threadPtr);
    COMPONENTPtr =vobjectPtr->getCOMPONENT();
    COMPONENTPtr->getIMPfile("R");
    break;
  case 4:
    threadPtr = firstThread (threadPtr, atoi(variationstr));
    vobjectPtr = threadPtr->version(atoi(versionstr));
    COMPONENTPtr = vobjectPtr->getCOMPONENT();
    COMPONENTPtr ->getIMPfile("R");
    break;
  default:
    cerr << "<ERROR: invalid number args for get vobject IMP>\n";
  }
}

void get_vobject_specfile_func(int number_arguments, char *proto_name,
                    char *operator_name,
                    char *variationstr, char *versionstr)
{
 TRACER("GET_vobject_specfile_func");
 V_OBJECT *vobjectPtr;
 threadPtr = findThread(proto_name, operator_name);
 if (!threadPtr)
  {
```

```
    return;
   }

  switch (number_arguments)
   {
   case 2:
    vobjectPtr = getDefaultVObject(threadPtr);
    COMPONENTPtr =vobjectPtr->getCOMPONENT();
    COMPONENTPtr->getSPECfile("R");
    break;
   case 4:
    {
     threadPtr = firstThread (threadPtr, atoi(variationstr));
     vobjectPtr = threadPtr->version(atoi(versionstr));
     COMPONENTPtr = vobjectPtr->getCOMPONENT();
     COMPONENTPtr ->getSPECfile("R");
     break;
    }
   default:
    cerr << "<ERROR: invalid number args for get vobject SPEC>\n";
   }
}

void get_vobject_sourcefile_func(int number_arguments, char *proto_name,
                        char *operator_name,
                            char *variationstr, char *versionstr)
{
 TRACER("GET_vobject_sourcefile_func");
 V_OBJECT *vobjectPtr;
 threadPtr = findThread(proto_name, operator_name);
 if (!threadPtr)
  {
   return;
  }

 switch (number_arguments)
  {
  case 2:
   vobjectPtr = getDefaultVObject(threadPtr);
   COMPONENTPtr =vobjectPtr->getCOMPONENT();
   COMPONENTPtr->getSOURCEfile("R");
   break;
  case 4:
   {
```

```
    threadPtr = firstThread (threadPtr, atoi(variationstr));
    V_OBJECT *vobjectPtr;
    vobjectPtr = threadPtr->version(atoi(versionstr));
    COMPONENTPtr = vobjectPtr->getCOMPONENT();
    COMPONENTPtr->getSOURCEfile("R");
    break;
   }
  default:
   cerr << "<ERROR: invalid number args for get vobject SOURCE>\n";
  }
}

void dump_vobject_files_func(int number_arguments, char *proto_name,
                    char *operator_name,char *file_write_option,
                      char *variationstr, char *versionstr)
{
 TRACER("dump_vobject_files_func");
 Boolean file_operation_successful = FALSE;
 V_OBJECT *vobjectPtr;
 threadPtr = findThread(proto_name, operator_name);
 if (!threadPtr)
  {
   return;
  }

 switch (number_arguments)
  {
  case 3:
   vobjectPtr = getDefaultVObject(threadPtr);
   vobject_locktime = vobjectPtr->getLockTime();
   if (vobject_locktime>0)
    // lock was set..prevent "w" checkout
    {
     if (strcmp(file_write_option,"w")==0)
      // if attempting "W" -- change to "r"
      {
       cerr << "<ERROR: Module "
        << vobjectPtr->getNodeName() << " locked by : "
         << vobjectPtr->getWorker()
          << " Resetting w option to ro>\n";
       strcpy(file_write_option,"r");
      }
     else
      {
```

```
        cerr << "<Caution: " << vobjectPtr->getNodeName()
             << " is locked.> \n" << "Date Locked: "
              << ctime(&vobject_locktime)
               << "operator files checked out in romode\n";
      }
    }
   else
    cout << vobjectPtr->getNodeName() <<"\n";
    file_operation_successful =
        vobjectPtr -> checkoutCOMPONENTNode(file_write_option);
   if ((file_operation_successful) &&
      ((strcmp(file_write_option,"W")==0) || (strcmp(file_write_option,"w")==0)))
    {
     vobjectPtr -> setLock();              // set root lock
     vobjectPtr -> setWorker();
     vobjectPtr -> resetLastOpFalse();
     vobjectPtr->putObject();
    }
   if (!file_operation_successful)
    cerr << "<ERROR: checking out " << vobjectPtr ->getName() ;
   break;
  case 5:
   threadPtr = firstThread (threadPtr, atoi(variationstr));
   vobjectPtr = threadPtr->version(atoi(versionstr));
   vobject_locktime = vobjectPtr->getLockTime();
   if (vobject_locktime>0)              // prevent checkout
    {
     if (strcmp(file_write_option,"w")==0)   // change "w" to "r"
      {
       cerr << "<ERROR: Module " << vobjectPtr->getNodeName()
        << " locked by : " << vobjectPtr->getWorker()
         << " Resetting w option to ro>\n";
       strcpy(file_write_option,"r");
      }
     else
      cerr << "<Caution: " << vobjectPtr->getNodeName()
       << " is locked.> \n" << "Date Locked: "
        << ctime(&vobject_locktime)
         << "operator files checked out in ro mode\n";
    }
   else
    cout << vobjectPtr->getNodeName() <<"\n";
    COMPONENTPtr = vobjectPtr->getCOMPONENT();
    file_operation_successful =
```

```
      vobjectPtr -> checkoutCOMPONENTNode(file_write_option);
     if ((file_operation_successful) &&
      ((strcmp(file_write_option,"W")==0) ||
        (strcmp(file_write_option,"w")==0)))
     {
      cerr << "Set lock, set worker, reset last op.\n";
      vobjectPtr -> setLock();                // set root lock
      vobjectPtr -> setWorker();
      vobjectPtr -> resetLastOpFalse();
      vobjectPtr->putObject();
      DDBControlData VControlFile;
      DDBControlData *VControlFilePtr = &VControlFile;
      char *Vnew_name = NULL;
      char *VV_name = vobjectPtr-> getName();
      VV_name[strlen(VV_name)-2] = '\0';
      char *VDDBControlpath =
          new char[strlen(dirNamePtr) + strlen(VV_name) + 13];
      strcpy(VDDBControlpath, dirNamePtr);
      strcat(VDDBControlpath, "/");
      strcat(VDDBControlpath, "ddbCtrlData.");
      strcat(VDDBControlpath, VV_name);
      VControlFilePtr->variation = vobjectPtr->getVariationNumber();
      VControlFilePtr->version = vobjectPtr->getVersionNumber();
      putddbControlFile(VDDBControlpath, VControlFilePtr);
     }
    if (!file_operation_successful)
     cerr << "<ERROR: checking out " << vobjectPtr ->getName() ;
    break;
   default:
    cerr << "<ERROR: invalid number args for get vobject FILES>\n";
   }
}

void dump_vobject_tree_func(int number_arguments, char *proto_name,
                    char *operator_name,char *file_write_option,
                    char *variationstr, char *versionstr)
{
 TRACER("dump_vobject_tree_func");
 Boolean file_operation_successful = FALSE;
 V_OBJECT *vobjectPtr;
 threadPtr = findThread(proto_name, operator_name);
 if (!threadPtr)
  {
   return;
```

```
}
switch (number_arguments)
 {
 case 3:
  vobjectPtr = getDefaultVObject(threadPtr);
  vobject_locktime = vobjectPtr->getLockTime();
  if (vobject_locktime>0)                // prevent checkout
   {
    if (strcmp(file_write_option,"w")==0)   // change "w" to "r")
     {
      cerr << "<ERROR: Module " << vobjectPtr->getNodeName()
        << " locked by : " << vobjectPtr->getWorker()
          << " Resetting write option to display>\n";
       strcpy(file_write_option,"r");
     }
    cerr << "<Caution: " << vobjectPtr->getNodeName()
         << " is locked.> \n" << "Date Locked: "
         << ctime(&vobject_locktime)
         << "Subtree checked out to display mode.\n";
   }
  else
   cout << vobjectPtr->getNodeName() << " "
      <<  vobjectPtr-> getVariationNumber()
      << " " << vobjectPtr->getVersionNumber() <<"\n\n";
  file_operation_successful =
   vobjectPtr -> checkoutCOMPONENTNode(file_write_option);
  if ((file_operation_successful) &&
    ((strcmp(file_write_option,"w")==0) ||
      (strcmp(file_write_option,"W")==0)))
   {
    vobjectPtr -> setLock();              // set root lock
    vobjectPtr -> setWorker();
    vobjectPtr -> resetLastOpFalse();
    vobjectPtr->putObject();
   }
  if (file_operation_successful)
   {
    vobjectPtr -> dumpSubtree(file_write_option);
    int vnum = vobjectPtr -> getVersionNumber();
    threadPtr -> putObject();
   }
   // dump rest of tree
  else
   {
```

```
    cerr << "<ERROR: checking out " << vobjectPtr ->getName()
      << " Aborting dump_vobject_tree_func>\n";
       TRACE("file_operation not successful");
   }
  break;
case 5:
  TRACE("case5");
  threadPtr = firstThread (threadPtr, atoi(variationstr));
  vobjectPtr = threadPtr->version(atoi(versionstr));
  vobject_locktime = vobjectPtr->getLockTime();
  if (vobject_locktime>0)                // prevent checkout
   {
    if (strcmp(file_write_option,"w")==0)   // change "w" to "r")
     {
      cerr << "<ERROR: Module " << vobjectPtr->getNodeName()
       << " locked by : " << vobjectPtr->getWorker()
        << " Resetting write option to display>\n";
          strcpy(file_write_option,"r");
     }
      cerr << "<Caution: " << vobjectPtr->getNodeName()
       << " is locked.> \n" << "Date Locked: "
       << ctime(&vobject_locktime)
       << "Subtree checked out to display mode.\n";
   }
  else
   cout << vobjectPtr->getNodeName() << " "
      <<  vobjectPtr-> getVariationNumber()
      << " " << vobjectPtr->getVersionNumber() <<"\n\n";
  file_operation_successful =
    vobjectPtr -> checkoutCOMPONENTNode(file_write_option);
  if ((file_operation_successful) &&
    ((strcmp(file_write_option,"w")==0) ||
      (strcmp(file_write_option,"W")==0)))
   {
    vobjectPtr -> setLock();              // set root lock
    vobjectPtr -> setWorker();
    vobjectPtr -> resetLastOpFalse();
    vobjectPtr->putObject();
   }
 if (file_operation_successful)
   {
    vobjectPtr -> dumpSubtree(file_write_option);
    int vnum = vobjectPtr -> getVersionNumber();
    threadPtr -> putObject();
```

```
    }
    // dump rest of tree
   else
    {
      cerr << "<ERROR: checking out " << vobjectPtr ->getName()
          << " Aborting dump_vobject_tree_func>\n";
      TRACE("file_operation not successful");
    }
   break;
  default:
   cerr << "<ERROR: invalid number args for get vobject TREE FILES>\n";
  }
}

void long_list_operators_func(int number_arguments, char *proto_name,
                    char *operator_name, char * variationstr,
                    char *versionstr)
{
 TRACER("long_list_operators_func");
 V_OBJECT *vobjectPtr;
 threadPtr = findThread(proto_name, operator_name);
 if (!threadPtr)
  {
   return;
  }

 switch (number_arguments)
  {
  case 2:
   TRACE("case2");
   vobjectPtr = getDefaultVObject(threadPtr);
   if (vobjectPtr)
    {
     TRACE("vobjectPtr not null");
     vobjectPtr->longlistOperatorNames();
    }
   break;
  case 4:
   TRACE("case4");
   TRACE("threadPtr not null");
   threadPtr = firstThread (threadPtr, atoi(variationstr));
   vobjectPtr = threadPtr->version(atoi(versionstr));
   // change made bj LJW 4/6/92
   if (vobjectPtr)
```

```
    {
     TRACE("vobjectPtr not null");
     vobjectPtr->longlistOperatorNames();
    }
   break;
  default:
   TRACE("default");
   cerr << "<ERROR: invalid number args for long list operators>\n";
  }
}

void long_list_children_func(int number_arguments, char *proto_name,
                    char *operator_name,char *versionstr)
{
 TRACER("long_list_children_func");
 V_OBJECT *vobjectPtr;
 threadPtr = findThread(proto_name, operator_name);
 if (!threadPtr)
  {
   return;
  }
 switch (number_arguments)
  {
  case 2:
     vobjectPtr = threadPtr->current();
     vobjectPtr -> listChildren();
   break;
  case 3:
     vobjectPtr = threadPtr->version(atoi(versionstr));
     vobjectPtr->listChildren();
   break;
  default:
   cerr << "<ERROR: invalid number args for long list children >\n";
  }
}

void long_list_parents_func(int number_arguments, char *proto_name,
                   char *operator_name, char *variationstr,
                     char *versionstr)
{
 TRACER("long_list_parents_func");

 V_OBJECT *parentPtr;
 V_OBJECT *vobjectPtr;
```

```
threadPtr = findThread(proto_name, operator_name);
if (!threadPtr)
 {
  return;
 }

switch (number_arguments)
  {
  case 2:
    TRACE("case2");
    vobjectPtr = getDefaultVObject(threadPtr);
    parentPtr = vobjectPtr->getParent();
    if (parentPtr)
      {
        V_OBJECT *grandparent = parentPtr->getParent();
        if (grandparent)
          grandparent -> listChildren();
        else
          cout << " <\nRoot Node: "
            << parentPtr->getNodeName() << ">\n";
      }
    break;
  case 4:
    TRACE("case4");
    threadPtr = firstThread (threadPtr, atoi(variationstr));
    vobjectPtr = threadPtr->version(atoi(versionstr));
    parentPtr = vobjectPtr->getParent();
    if (parentPtr)
      {
        V_OBJECT *grandparent = parentPtr->getParent();
        if (grandparent)
          grandparent -> listChildren();
        else
          cerr << "<Can not list Parent/siblings of Root V_Object>\n";
      }
    else
      cerr << "<Currently located at Root V_OBJECT>\n";
    break;
  default:
    TRACE("dfault");
    cout << "<ERROR: invalid number args for long list children >\n";
  }
}
```

```
DDBControlData* getddbControlFile (char *ddbctrlfile)
      {
      TRACER("GETddbControlFile");
      DDBControlData *VarVerNumberPtr = new DDBControlData;
      fstream infile;
      infile.open(ddbctrlfile,ios :: in | ios :: nocreate);
      int result = infile.good();

      if (!result)
           {
           infile.close();
           return VarVerNumberPtr = (DDBControlData *)0;
           }
      infile >> VarVerNumberPtr-> variation;
      infile >> VarVerNumberPtr-> version;
      infile.close();
      return  VarVerNumberPtr;
      }

void putddbControlFile (char *ddbctrlfile, DDBControlData *DDBCtrl)
      {
      fstream outfile;
      outfile.open(ddbctrlfile,ios :: out);
      outfile << DDBCtrl-> variation << "\n";
      outfile << DDBCtrl-> version << "\n";
      outfile.close();
      }

void add_new_variation_func(int number_arguments, char *proto_name,
                            char *operator_name,
                            char *variationstr,
                            char *versionstr)
{
 TRACER("add_new_variation_func");
 char *prototype_name = (char*)(My_String(proto_name) +
      My_String(PROTOTYPE_EXT));

    V_OBJECT *variationPtr = (V_OBJECT *)0;
    V_OBJECT *NewVariationPtr = (V_OBJECT *)0;
    V_OBJECT *versionPtr  = (V_OBJECT *)0;
    V_OBJECT *new_parent = (V_OBJECT *)0;
    THREAD *TheThread = (THREAD *)0;

    prototypeptr = (PROTOTYPE*)OC_lookup(prototype_name);
```

```
  if (prototypeptr)
   {
    cout << operator_name << "\n";
    char *p_operator_name = buildThreadName(operator_name, 1);
    threadPtr = (THREAD *)OC_lookup(p_operator_name);
   }
  else
   {
    cerr << "<ERROR: getting Prototype in ADD_NEW_VARIATION:>\n";
    return;
   }

switch (number_arguments)
 {
  case 2:
   {
    if (!threadPtr)
      {
        variationstr = "1";
        versionstr = "1";
      }
    else
      {
        // do a get control data here to check if a control file exists.
        V_OBJECT *default_Version = getDefaultVObject(threadPtr);
        new_parent = default_Version->getParent();
      }
    break;
   }
  case 4:
   {
    if (threadPtr)
      {
       variationPtr= threadPtr -> version(1);
       NewVariationPtr = variationPtr -> variation(atoi(variationstr));
       TheThread = NewVariationPtr -> getThread();
         if (TheThread)
           {
             versionPtr = TheThread -> version(atoi(versionstr));
             new_parent = versionPtr->getParent();
           }
      }
    break;
   }
```

```
    default :
     {
      cerr << "<ERROR: invalid number args for add new variation>\n";
      return;
     }
  }
  DIRECTORY *capsdirectory;
  capsdirectory = new DIRECTORY();
  capsdirectory->read_directory(operator_name);
  capsdirectory->updatetimestamp();
  TREENODE_linkedlist operatorList = capsdirectory->getOperatorList();
  TREENODE *rootnode = capsdirectory->find_treenode(operator_name);
  TREENODE *tree_root = new TREENODE(rootnode,NULL);
  TREE *workingtree = new TREE(tree_root, operator_name);
  workingtree->build_tree(tree_root,operatorList);
  V_OBJECT *new_root = tree_root->checkin_node(new_parent);
  if (!new_root)
   {
    cerr << "<ERROR: Could not establish new_root in"
       << "add_vobject_and_subtree_func. Aborting.>\n";
       return;
   }
  new_root->setNodeName(tree_root->getname());
  tree_root->checkin_subtree(new_root);
/*Skip release lock of first variation and version checked into database
 since it does not exist yet. */
/*
  if (!((atoi(variationstr) == 1) && (atoi(versionstr) == 1)))
     {
      release_subtree_lock_func( 4, proto_name, operator_name,
                        variationstr, versionstr);
     }
*/
}

THREAD* firstThread (THREAD *threadPtr, int variationint)
     {
      V_OBJECT *variationPtr = threadPtr -> version(1);
      V_OBJECT *NewVariationPtr = variationPtr -> variation(variationint);
      THREAD *firstthreadPtr = NewVariationPtr -> getThread();
      return firstthreadPtr;
     }

V_OBJECT* getDefaultVObject(THREAD *initialThreadPtr)
```

```
    {
     V_OBJECT *default_Version = initialThreadPtr-> version(1);
     int latest = default_Version-> getNumberOfVariations();
     V_OBJECT *working_default_Version =
              default_Version-> variation(latest);
     initialThreadPtr =  working_default_Version->getThread();
     default_Version = initialThreadPtr->current();
     cerr << "Default VOBJECT being being accessed. Variation: " << latest
         << " Version: " << default_Version-> getVersionNumber()
         << "\n";
     return default_Version;
    }

THREAD* findThread(char *proto_name, char *operator_name)
    {
     char *prototype_name = (char*)(My_String(proto_name) +
     My_String(PROTOTYPE_EXT));

     prototypeptr = (PROTOTYPE*)OC_lookup(prototype_name);
     THREAD *TempthreadPtr = (THREAD *)0;

     if (prototypeptr)
       {
        char *p_operator_name = buildThreadName(operator_name, 1);
        TempthreadPtr = (THREAD *)OC_lookup(p_operator_name);
        if (!TempthreadPtr)
          {
           cerr << "<ERROR: Invalid VObject name. Can not get Thread.>\n";
          }
       }
     else
       {
        cerr << "<ERROR: Invalid Prototype name.>\n";
       }
     return TempthreadPtr;
    }
```

LIST OF REFERENCES

[Ref. 1] Douglas, Bryant, S., *A Conceptional Level Design of a Design Database for a Computer-Aided Prototyping System*, Master's Thesis, Naval Postgraduate School, Monterey, California, March 1989.

[Ref. 2] Ketabchi, M. A., Berzins. V., March, S.,"ODM: Object Oriented Data Model for Design Databases", *Proc. ACM. Computer Science Conference*, Feb 1986, pp. 261-269.

[Ref. 3] Ketabchi, M. A., Berzins. V., "Modeling and Managing CAD Databases, "*IEEE Computer*, pp. 46-49, February 1987.

[Ref. 4] Luqi, "Software Evolution Through Rapid Prototyping", Computer, v.22,no.5, pp.13-25, May 1989.

[Ref. 5] Katz, Randy, H., Chang, Ellis, Bhateja, Rajiv, "Version Modeling Concepts for Computer-Aided Design Databases", *ACM, SIGMOD Rec. 15*, pp. 397-386, April 1, 1986.

[Ref. 6] McKenna, J., "Teaching OOP", *OOPSLA '88 Conference Proceedings*, The Association for Computing Machinery, New York, NY, 1988.

[Ref. 7] White, Laura, J., *The Development of a Rapid Prototyping System*, Master's Thesis, Naval Postgraduate School, Monterey, California, December 1989.

[Ref. 8] Luqi, *A Graph Model for Software Evolution*, IEEE Transactions on Software Evolution, v. 16, n. 8. pp. 917-927, August 1990.

[Ref. 9] Berzins, V., *Advanced Software Engineering* (Course Notes), 1991.

[Ref. 10] Lewis, Gary, W., Dwyer, Andrew, P., *The Development of a Design Database for the Computer Aided Prototyping System*, Master's Thesis, Naval Postgraduate School, Monterey, California, September 1991.

[Ref. 11] Ontologic Inc., *ONTOS, Object Database Documentation Set, Release 2.0,* Burlington, MA.

[Ref. 12] Yourdon, E., *Modern Structured Analysis*, Yourdon Press, 1989.

[Ref. 13] Frakes, William, B., Fox, Christopher, J., Nejmeh, Brian, A., *Software Engineering in the UINIX/C Environment*. Englewood Cliffs, N.J.:Prentice Hall, 1991.

[Ref. 14] Perry, D., "The Inscape Environment," in *Proc. 11th Int. Conf. Software Engineering*, IEEE, 1989, pp. 2 -12.

[Ref. 15] Kaiser. G., Feiler, P., and Reps, T., "Intelligent Assistance for software development and maintenance," *IEEE Software*, pp. 40_49, May 1988.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center — 2
   Cameron Station
   Alexandria, VA 22304-6145

2. Dudley Knox Library — 2
   Code 52
   Naval Postgraduate School
   Monterey, CA 93943

3. Computer Science Department — 2
   Code CS
   Naval Postgraduate School
   Monterey, CA 93943

4. Office of the Assistant Secretary of the Navy — 1
   Research Development and Acquisition
   Department of the Navy
   Attn: Mr. Gerald A. Cann
   Washington, DC 20380-1000

5. Office of the Chief of Naval Operations — 1
   OP-094
   Department of the Navy
   Attn: VADM J. O. Tuttle, USN
   Washington, DC 20301-3040

6. Director of Defense Information — 1
   Office of the Assistant Secretary of Defense
   (Command, Control, Communications, & Intelligence)
   Attn: Mr. Paul Strassmann
   Washington, DC 20301-0208

7. Center for Naval Analysis — 1
   4401 Ford Avenue
   Alexandria, VA 22302-0268

8. Director of Research Administration 1
   Attn: Prof. Howard
   Code 08Hk
   Naval Postgraduate School
   Monterey, CA 93943

9. Chairman, Code CS 1
   Computer Science Department
   Naval Postgraduate School
   Monterey, CA 93943-5100

10. Prof. Luqi, Code CSLq 10
    Computer Science Department
    Naval Postgraduate School
    Monterey, CA 93943

11. Chief of Naval Research 1
    Attn: ADM. Miller
    800 N. Quincy Street
    Arlington, VA 22217

12. Director, Ada Joint Program Office 1
    OUSDRE (R&AT)
    Room 3E114, The Pentagon
    Attn: Dr. John P. Solomond
    Washington, DC 20301-0208

13. Carnegie Mellon University 1
    Software Engineering Institute
    Attn: Dr. Dan Berry
    Pittsburgh, PA 15260

14. Office of Naval Technology (ONT) 1
    Code 227
    Attn: Dr. Elizabeth Wald
    800 N. Quincy St.
    Arlington, VA 22217-5000

15. Defense Advanced Research Projects Agency (DARPA) 1
Integrated Strategic Technology Office (ISTO)
Attn: Dr. B. Boehm
1400 Wilson Boulevard
Arlington, VA 22209-2308

16. Defense Advanced Research Projects Agency (DARPA) 1
ISTO
1400 Wilson Boulevard
Attn: LCol Eric Mattala
Arlington, VA 2209-2308

17. Defense Advanced Research Projects Agency (DARPA) 1
Director, Tactical Technology Office
1400 Wilson Boulevard
Arlington, VA 2209-2308

18. National Science Foundation 1
Division of Computer and Computation Research
Attn: K. C. Tai
Washington, DC 20550

19. Commander Space and Naval Warfare Systems Command 1
SPAWAR 3212
Department of the Navy
Attn: Cdr M. Romeo
Washington, DC 20363-5100

20. Office of Naval Research 1
Computer Science Division, Code 1133
Attn: Dr. Gary Koob
800 N. Quincy Street
Arlington, VA 22217-5000

21. Office of Naval Research 1
Computer Science Division, Code 1133
Attn: Dr. A. M. Van Tilborg
800 N. Quincy Street
Arlington, VA 22217-5000

22. Office of Naval Research 1
Computer Science Division, Code 1133
Attn: Dr. R. Wachter
800 N. Quincy Street
Arlington, VA 22217-5000

23. University of CA at Berkeley 1
Department of Electrical Engineering and
Computer Science
Computer Science Division
Attn: Dr. C.V. Ramamoorthy
Berkeley, CA 90024

24. University of MD 1
College of Business Management
Tydings Hall, Room 0137
Attn: Dr. Alan Hevner
College Park, MD 20742

25. University of MD 1
Computer Science Department
Attn: Dr. N. Roussapoulos
College Park, MD 20742

26. University of Massachusetts 1
Department of Computer and Information Science
Attn: Dr. John A. Stankovic
Amherst, MA 01003

27. University of Pittsburgh 1
Department of Computer Science
Attn: Dr. Alfs Berztiss
Pittsburgh, PA 15260

28. Commander, Naval Surface Warfare Center, 1
Code U-33
Attn: Dr. Philip Hwang
10901 New Hampshire Avenue
Silver Spring, MD 20903-5000

29. Attn: Joel Trimble 1
1211 South Fern Street, C107
Arlington, VA 22202

30. United States Laboratory Command 1
Army Research Office
Attn: Dr. David Hislop
P. O. Box 12211
Research Triangle Park, NC 27709-2211

31. Persistent Data Systems 1
75 W. Chapel Ridge Road
Attn: Dr. John Nester
Pittsburgh, PA 15238

32. Commandant of the Marine Corps 1
Ada Joint Program Representative
Code CCI
Attn: Capt Gerald Depasquale
Washington, DC 20301

33. Commandant of the Marine Corps 1
Code TE-06
Washington, DC 20301

34. Commandant of the Marine Corps 1
Code CCT-60
Attn: Ltcol L. Machabee
Washington, DC 20301

35. Mrs. Loretta O'Loughlin 1
46 Willow Road
Metuchen, NJ 08840

36. Lt. Greg Hammond USN 1
5 Knoll Court
Hercules CA.

37. Capt. Michael D. O'Loughlin USMC 1
18 Continental Road
Summerset NJ. 08873